

تحليل وتصميم الخوارزميات

Algorithms Design and Analysis

تأليف

الدكتور	الأستاذ المساعد	الباحث
حسن ياسين طعنه	هند رستم محمد شعبان	حسن ثابت رشيد كرماشة

hind_restem@yahoo.com
hassan_thabit@yahoo.com

تحليل وتصميم الخوارزميات
Algorithms Design and Analysis

الفهرس

الفصل الأول: مقدمة (Introduction)

- 1-1: مقدمة في الخوارزميات (Algorithms Introduction)
- 2-1: كيفية تحليل الخوارزمية (Algorithm Analysis)
- 3-1: الوقت الكلي لتنفيذ الخوارزمية (Execution Time)
- 4-1: الحالات الأفضل والأسوأ والمتوسطة للتحليل (Best & Worst & Average Casse Analysis)
- 5-1: الصيغ التقريبية (Asymptotic notation)
- 6-1: الصيغ الشائعة لأوقات التنفيذ (The Times of Executive Notation)
- 7-1: الاستدعاء الذاتي لشجرة التسيطات أو الاستدعاءات (Recursion) (Tree)
- 8-1: قياس الانجازية (Performance Measurement)

الفصل الثاني: الترتيب (Sorting)

- 1-2: خوارزميات الترتيب (Sorting Algorithms)
- 2-2: أنواع الترتيب (Types of Sorting)
- 3-2: خوارزميات الترتيب الداخلي (Internal Sort Algorithms)
 - 1- ترتيب الاختيار (Selection Sort)
 - 2- ترتيب الفقاعي (Bubble Sort)
 - 3- ترتيب الإضافة (Insertion Sort)
 - 4- ترتيب شيل (Shell Sort)
 - 5- الترتيب السريع (Quick Sort)
 - 6- ترتيب الأساس (Radix Sort)
 - 7- ترتيب المؤشرات (Pointers Sort)
 - 8- ترتيب الشجري لشجرة البحث الثنائية (Tree Sort)
 - 9- Topological sorting
- 4-2: خوارزميات الترتيب الخارجي (External Sort Algorithms)
 - 1- ترتيب الدمج (merge Sort)
 - 2- ترتيب الدمج المتوازن ذو الممرتين (Balanced Two-Way Merge Sort)

الفصل الثالث: البحث (Searching)

- 1-3: البحث (Searching)
- 2-3: البحث التسلسلي (Sequential Search)
- 3-3: البحث الثنائي (Binary Search Algorithm)
- 4-3: البحث في الشجرة الثنائية (Binary Search tree)
- 5-3: تعقيد خوارزمية البحث (Algorithm search Complexity)

الفصل الرابع: الأمثلية في مسائل تصميم الخوارزميات (Optimization in Algorithms Design Equations)

- 1-4: المخططات (Graphs)
- 2-4: أنواع المخططات (Type of Graphs)
- 3-4: طول المسار (Path Length)
- 4-4: طريقة الجشع أو الطماع (Greedy Method)
- 5-4: مسألة الجراب (Knapsack Problem)
- 6-4: استخدام قاعدة الطماع في إيجاد أمثلية البيانات

الفصل الخامس: البرمجة الديناميكية (Dynamic Programming)

- 1-5: البرمجة الديناميكية (Dynamic programming)
- 2-5: أمثلة على البرمجة الديناميكية
- 3-5: تجميع البيانات (Data clustering)
- 4-5: خوارزمية (Dijkstra)
- 5-5: أمثلة لتطبيق خوارزمية (Dijkstra)
- 6-5: المخططات المتعددة المراحل (Multistage graph)
- 1-6-5: الطريقة التصاعدية (Forward approach)
- 2-6-5: الطريقة التناقصية (Backward approach)
- 3-6-5: طريقة اقتفاء الأثر رجوعاً (Back Tracking)

الفصل الأول

مقدمة

(Introduction)

1-1: مقدمة في الخوارزميات (Algorithms Introduction) :

- الخوارزمية هي مجموعة محددة من التعليمات (خطوات الحل) التي تؤدي إلى إنجاز وظيفة (مهمة) معينة ويجب أن تتوافق فيها الشروط التالية :
1. المدخلات (Input) : صفر أو أكثر من القيم .
 2. المخرجات (Output) : قيمة واحدة على الأقل .
 3. الوضوح (Definiteness) : كل خطوة فيها (الخوارزمية) واضحة المعاني وغير غامضة أي يجب أن نفهم من قبل جميع الناس (مفهوم الحسابات).
 - و على سبيل المثال نأخذ العبارة "Add 6 or 7 to X" هذه العبارة غير مفهومة بها في الخوارزمية لأنها عبارة غير واضحة .
 4. المحدودية (Finiteness) : كل خطوات الخوارزمية يمكن حلها في فترة زمنية محددة ، والتوضيح تلك نأخذ العبارة " قسم الرقم (10) على (3) بدقة عالية (كاملة) " هذه العبارة غير محدودة ويجب أن لا يسمح بها دخال البرنامج .
 5. الفعالية (Effectiveness) : كل خطوة تكون ممكنة الحل أو التنفيذية ، مثال تلك العبارة " $0/3$ " لا يمكن حلها أبداً .

يمكن لنا أن نوضح الفرق بين الخوارزمية والبرنامج حيث أنه في النظرية الاحتمالية يوجد فرق بين الخوارزمية والبرنامج ، ففي الخوارزمية يجب أن تتوافق الشروط الخمسة الأتفة للذكر ويمكن وصفها بطرق عديدة مثل لغة طبيعية مع التأكيد على شرط الوضوح ، لغة خوارزمية (Pseudo-code) ، مخططات انسيابية (Flow chart) ، بينما يمكن في البرنامج عدم تحقق الشرط الرابع حيث إن نظام التشغيل هنا هو الذي يعتمد على البرنامج ويوصف البرنامج بلغة الحاسبة حيث أنه يصمم ليتحكم في تنفيذ مجموعة من الأعمال (Jobs) بحيث عند عدم توفر عمل معين فإنه لا ينتهي من أعماله بل يستقر ويدخل في حالة لتتظار لحين إدخال عمل جديد .

إن لكل لغة برمجية يوجد مترجم أو مفسر ولا يمكن تواجدهما معاً حيث إن المفسر يقوم بتنفيذ البرنامج خطوة خطوة (step by step) بينما المترجم فإنه يتخذ البرنامج كاملاً ويظهر النتائج والأخطاء ، هذا يعني إن البرنامج هو عبارة عن خوارزمية وهيكل يأتى أي أنه طريقة لتنظيم البيانات.

خطوات تطوير البرنامج :

تتم عملية تطوير البرنامج بخمس خطوات رئيسية هي :

1. توصيف المتطلبات (Requirement specification):

هو تحديد المدخلات والمخرجات.

2. التصميم (Design):

هو تحديد العمليات الرئيسية التي تطبق على كل كيان بياني والفقرات وجرد أجهزة معالجة لتنفيذ هذه العمليات.

3. التحليل (Analysis):

هو المفصلة بين الخوارزميات المعروفة التي تحل نفس المسألة تبعاً لمقاييس المفصلة متقناً عليها (تقنيات الوقت، تقنيات الفراغ (الخرن)) باختيار أفضلها .

4. التحسين والتشفير (Refinement & Coding):

في هذه الخطوة يتم تحديد التمثيل الباقي لكل كيان ثم كتابة الإجراءات لكل عملية على تلك الكيفيات وتكون نسخة متكاملة للبرنامج.

ملاحظة // التحليل يصلح الأخطاء اعتماداً على تقنيات الخرن والوقت بينما التحسين يصلح الأخطاء اعتماداً على النتائج الظاهرة في نهاية البرنامج.

5. التحقق من الصلابة (Verification):

تضمن هذه الخطوة ثلاث جوانب هي :

أ- البرهنة على الصحة (Proving) :

قبل استخدام البرنامج يجب إثبات أنه صحيح حيث يتم استخدام الطرق المعروفة للبرهنة على الصحة.

ب- الاختبار (Testing):

هي عملية توليد نماذج بيانية يعمل عليها البرنامج حيث إن الهدف منها هو إعطاء إشارة على وجود أخطاء في البرنامج.

ج- تشخيص الأخطاء (Debugging):

عملية تحديد مواقع الأخطاء البرمجية في البرنامج وتصحيحها .

ملاحظة : إن التعريف يختلف عن الصلابة فالترقيق هو معرفة شيء قد يكون صحيح أو خطأ بينما الصلابة هي معرفة شيء يجب أن يكون صحيح .

أما النموذج فهو تحقيق تمثيل بياني بالشكل الصحيح.

في علم الحاسبات يتم أولاً الاختبار وبعدها يتم البرهنة بينما في علم الرياضيات يتم البرهان وبعده الاختبار.

2-1: كيفية تحليل الخوارزمية (Algorithm Analysis)

تحليل الخوارزمية هو تحديد الكفاءة للخوارزمية ومن ثم تصنيفها حيث يوجد مقياسين مرتبطين مباشرة بشجرة الخوارزمية هما :

1 - مقياس تعقيدات الفراغ أو التخزين (Space Complexity): هي كمية الذاكرة التي يتطلبها تشغيل البرنامج حتى اكتماله بحيث يعتمد هذا النوع على جزئين :

أ- جزء ثابت : هو مستقل عن خصائص المدخلات والمخرجات حيث يتضمن هذا الجزء فراغ التعليمات (Code space) ، الفراغ المخصص للمتغيرات (Data Space) سواء كانت البسيطة أو المتغيرات المركبة ذات الحجم الثابت إضافة إلى فراغ التراكيب ، الخ .
 ب- جزء متغير: يتألف من الفراغ الذي يتطلبه البرنامج بالمتغيرات المركبة والتي يعتمد حجمها على مثال المسألة المراد حلها ، إضافة إلى فراغ المكدس المستخدم في التداخل (Reaction).



شكل (1): تحليل الخوارزمية

إن التخزين التبادلي يمكن توضيحه بالمتغيرات التي يدخلها البرنامج (أي يدخل قبتها) وكذلك يتحكم بأسمائها فهي متعدة على إدخال البرنامج.

أما القيم المركبة فهي المصفوفة التي تمثل بالمكدس حيث المؤشر هو (Sip) عمادياً وبرمجياً يسمى (Top) ، وفيما يخص تصنيف الخوارزميات فإن المهم هنا هو مستوى المصفوفة أي المكدس.

يمكن صياغة تعقيدات الخزن للبرنامج كالآتي :

ثابت

Code segment
Data segment
Heap segment
Stack segment

إن جزء (Code Segment) يمكن تمثيله كما الحوال الجاهزة ، أما (Heap Segment) فيكون المتغيرات التي يستخدمها البرنامج .

وعليه يمكن صياغة تعقيدات الفراغ $S(p)$ للبرنامج (p)

$$S(p) = \text{Const} + Sp$$

حيث إن :

Const : تمثل جزء (Code segment) والمتغيرات البسيطة .

Sp : تمثل خصائص المثال .

2- تعقيدات الوقت (Time Complexity) : هي كمية الوقت التي يتطلبها تنفيذ البرنامج حتى اكتماله ويتألف من :

$$T(p) = \text{Const} + tp$$

حيث :

Const : يمثل ثابت خاص بوقت الترجمة أو التفسير .

Tp : يمثل وقت تشغيل البرنامج .

مثال 1 // لبيان تعقيدات الفراغ (الخزن) والوقت لدالة معينة (بلغة C++) :

```
Float abc(float a,float b,float c)
{return(a+b+5*c+(a+b+c)/(a+b)+4.0); }
```

تعقيدات الفراغ أو الخزن :

تطلب الدالة (abc) خمسة خلايا خزن لثمة قيم المتغيرات (a,b,c) والمتغير الذي يحمل اسم الدالة وعنوان العونة (Return address) وهو خزن ثابت لا يعتمد على خصائص المثال (a,b,c).

$$S_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا تعني إن الخزن ثابت أي لا يعتمد على خصائص المثال .

تعقيدات الوقت : تستخدم صيغة عد الخطوات (Steps Count) لقياس تقدير الوقت حيث إن عدد الخطوات لهذه الدالة يساوي واحد ، ولها فإن :

$$T_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا أيضا تعني إن الوقت ثابت .

مثال //2 اكتب خوارزمية لإيجاد القاسم المشترك الأعظم (Greatest Common Divisor) لعددين صحيحين .

//الحل

```

Step 0 : [ check m and n ]
    If m <= 0 or n <= 0 then
        Print error.
Step 1: [ test m and n ]
    If m < n then
        Inter change m by n.
Step 2: [find the remainder]
    Divid m by n and let r is
    Remainder we will have 0 <= r < n.
Step 3: [is r = zero]
    If r = 0 then
        GCD = n and exit.
Step 4:[ inter change ]
    M ← n, n ← r go to step 2

```

مثال تطبيق //1

m	n	r
10	6	4
6	4	2
4	2	0

GCD = 2

مثال تطبيق // 2

m	n	r
20	130	
130	20	10
20	10	0

GCD =10

عدد مرات تنفيذ العبارة (Frequency Count) :

إن عدد مرات تنفيذ العبارة يختلف حسب عينة البيانات . حيث يوجد لدينا ما يسمى بوقت التنفيذ المفرد للعبارة (execution time for single).

Total execution time = frequency count * execution time for single

إن الوقت الكلي لتنفيذ يعتمد على العوامل التالية :

1. نوع الحاسبة (Computer type).
2. لغة البرمجة (Programming language).
3. الوقت التنفيذي الخاص لكل عبارة (Total execution time).
4. نوع المترجم أو المفسر (Compiler and interpreter).

مثال 3// افترض وجود الأجزاء البرمجية الآتية مرقبة من 1 إلى 3 كالآتي :

مثال 1	$x = x + 1;$	$F_c = 1$
مثال 2	$\text{For}(\text{int } i=1; i \leq n; i++)$ $x = x + 1;$	$F_c = n$
مثال 3	$\text{For}(\text{int } i=1; i \leq n; i++)$ $\text{For}(\text{int } j=1; j \leq n; j++)$ $x = x + 1;$	$F_c = n^2$

لو افترضنا أن ($n = 10$) فإن مثال رقم (2) فيه عدد مرات تكرار الخطوة التنفيذية هو (10) وعدد المرات في المثال رقم (3) هو (100).
نستنتج من ذلك أن المثال رقم (1) ينفذ أسرع من المثالين (2) و (3) ومثال (2) أسرع من مثال (3).

3-1: الوقت الكلي لتنفيذ الخوارزمية (Execution Time):

تنظيم الترتيب للخوارزمية (Order of magnitude of Algorithm) :
هو مجموع تكرارات جميع العبارات التنفيذية التي يسويها يحدد التقدير المسبق لوقت تنفيذ الخوارزمية.
إن العبارة الغير تنفيذية تعني العبارة التي ليس لها تأثير على البرنامج مثل عبارة التعليق في أي لغة برمجية يمكن استخدامها.

مثال// لدينا مصفوفة A بحجم n ، نحسب مجموع كل صف ونحزن قيمته في مصفوفة اسمها S ، ثم نحسب المجموع الكلي لعناصر المصفوفة S ، ثم اعطي عدد تكرارات المرات .

$$\text{Sum} = \sum_{j=1}^n a_{ij}$$

الحل // توجد طريقتين:

```

1 Grandtotal = 0;
For(int k = 1; k <= n; k++)
{ s[k] = 0;
For(int j = 1; j <= n; j++)
{ s[k] = s[k] + a[k,j];
Grand total = (Grand total + s[k]);
}
}

```

نلاحظ ان عدد التكرارات يساوي $2n * n$

```

2- Grandtotal = 0;
For(int k = 1; k <= n; k++)
{ s[k] = 0;
For(int j = 1; j <= n; j++)
{ s[k] = s[k] + a[k,j]; }
Grand total = Grand total + s[k];
}

```

وهنا نلاحظ انها تساوي $n^2 + n$

مثال ٢: ما كلفة خوارزمية CN وخوارزمية ثنائية CN^2 عموماً إن C هي ثابت ، قم بعمل مقارنة بين الخوارزميتين من حيث وقت التنفيذ عموماً إن:

C الأولى = 10 ، والثانية = 0.5

و $n = \{ 1, 5, 10, 15, 20, 25, 30 \}$

n	CN	CN ²
1	10	0.5
5	50	12.5
10	100	50
15	150	112.5
20	200	200
25	250	312.5
30	300	450

ملحظة // إذا كانت قيمة n أقل أو تساوي 20 فإن وقت الخوارزمية الثانية أقل من وقت الخوارزمية الأولى أما بعد عن ذلك التكرار أو التحسينات أقل ، لكن بعد هذه النقطة (20) أي (25, 30) فإن وقت الخوارزمية الأولى يكون أقل ، هذه المرة (يسج العكس)

هدف اليوم بالحصة وقت التعداد للخوارزمية خطي هناك إننا نجد $(O(g(n)))$ وهذا يعني ان وقت تنفيذ دنا يستغرق أكثر من $(C * g(n))$ حيث إن (C) هو كمية ثابتة وإلا n تعقل عدد التعديلات المطلوبة لمحوط الخوارزمية لتختلف حصص m .

مثلاً:

- 1 $O(1)$ معنى ذلك أن وقت الحساب ثابت مثل $(x \leftarrow x+1)$ صغر البرنامج
- 2 $O(n)$ وهي أن وقت الاحتساب ذو طبيعة خطية مثل طباعة جميع عناصر مصفوفة حجمها n أو مثلاً إيجاد عنصر في قائمة موصولة
- 3 Quadratic (وقت تربيع $O(n^2)$) مثل وقت ترتيب عناصر قائمة باستخدام عناصر قسمة متعاقبة .
- 4 $O(n^3)$ في الوقت الترتيب لحمل عناصر مصفوفة $(n \times n \times n = 0)$.
- 5 $O(2^n)$ يعني أن لوقت الترتيب لحمل جميع عناصر مصفوفة مثلاً مساريًا للعنصر هو استخدام الصيغة الأسية .
- 6 $O(\log(n))$ في الصيغة مثل وقت البحث باستخدام صيغة التوابع ثنائية مثل تلك للوصول إلى عقدة معينة في شجرة ثنائية.

ملاحظة/ قيمة \log دائماً تكون مقصورة بين (0) و (1) أي أقسام عشرية .

مثال // هناك اقيم لتاليه $n = \{1, 2, 4, 8, 16\}$
 مفرد بتضمينها على نحو زدهيفت الآليه وفازلتها بجداول تعرض توضيحها .

n	$\log_2 n$	$N \log_2 n$	N^2	N^3	2^n
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536

$$\log_2(x) \quad \log_{10}(x) \times 3.322$$

مربعين // المرح التوال الساعه ووضح عصفه الفرق بين الخو برين مفرد

- ملاحظة 1/ وقت $O(n)$ ووقت $O(n \log n)$ ينشأ كل منهما بصورة أخطأ من التوال الأخرى .
- ملاحظة 2/ عندما تكون حجم تسلسل كبير يصبح الخوارزمية لها بعد وقت كبير مثلاً الخوارزمية التي عدد عناصرها $O(n \log n)$ تكون على القيمة أو غير عصفه .
- ملاحظة 3/ نحو زدهيفت التي وقت تنفيذ بالصيغة الأسية دائماً يمكن اعتمادها فقط إذا كانت قيمة (n) صغيرة، حيث أن كانت قيمة n صغيرة فإن عند العصف قبل والتأني في وقت التعداد أسرع والتأني الأخير سيكون واضح .

مثال // هناك لساله الآليه

$$F = a + b^2 + c + (a + b) \quad (a + b) + 4(0)$$

عصاً إلى التوال $(a=2.0, b=3.0, c=0.0)$.
 المطلوب / تحديد تعقيدات الوقت وتحديد الحرج عصاً إلى (a, b, c) في قيم حقيقيه

ملاحظة: عند التعويض بالمتغير $\frac{2}{3} = 2.33$ ، الخزن يعطي 2.33 في المتغيرات الموجودة في

السؤال *

مصفوفات الحرج - يوجد لدينا مصفوفة حاليًا حرجية هي a, b, c وعكسها العكس F وتعتبر الاسم للصفة

، معنى ذلك إن $(c, b, a) = S$ أي لا يوجد جزء من S هو أي المتغير لك

في حالة إن تكون المصفوفات هذه الدالة n من العنصر ما هي مصفوفات الخزن هنا وعملها في

n=3

و (a, b, c) قيمها كالتالي.

a	b	c
3	2	1
2	3	2
1	4	1

للحل نقوم بالتالي -

نقوم بعمل جدول كالآتي - معنى ذلك أنه يوجد قيم صغيرة بالمصفوفات فالنتيجة لا يساوي صفر
ونعتمد على قيم المصفوفات

مصفوفات قوف

في الحالة الأولى وفي الدالة يساوي واحد (Count=1) لأنها تتغير مرة واحدة وهي الحالة الثانية
التي تعتمد على عدد المصفوفات (Count=n)

مثال: إيجاد مجموع عناصر مصفوفة بحدة الحد كفي في مصفوفة الدالة

$$Sum = \sum_{i=1}^n a_i$$

Float Sum(float a[], int n)

{ float S=0.0 // (عدد المصفوفات = 1)

For(int i=1; i<=n; i++) // (عدد المصفوفات = n)

S+=a[i]; // (n)

Return S // (1)

}

يقسم مثال المسألة بالمتغير (n)

مصفوفات الحرج - نطلب الدالة (Sum) متعة حاليًا حرجية لحرج قيم (I, S, n) وعنوان المصفوفة

[a والسفر التي تجعل اسم الدالة المتكامل في عنوان الدالة]

وهو خزن ثابت لا يعتمد على حصة المتغير (أي لا يعتمد على تغير قيمه n)

$$S_{sum}(n)=0$$

تعريف الوقت

$$T_{sum}(n)=2n+3$$

لاحظ إن العلاقة التي تربط الوقت بعدد العناصر هي علاقة خطية وهي أفضل من التربيعية
مثلاً، سوف نجد نفس العدد السابق ولكن هذه المرة بطريقة الاستدعاء الذاتي (Recursion).

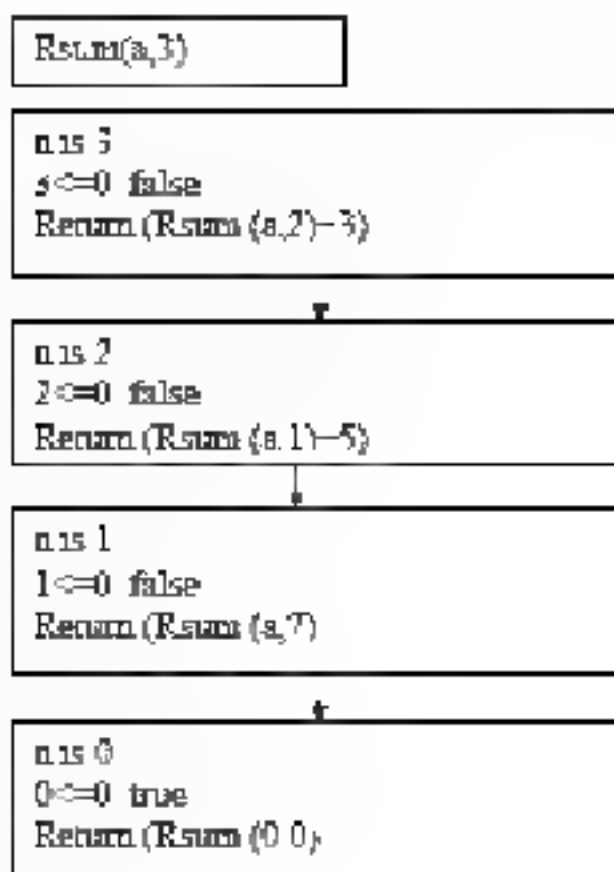
$$Sum = \sum_{i=1}^n a_i$$

$$Rsum(a, n) = \begin{cases} 0.0 & \text{if } n \leq 0 \\ Rsum(a, n-1) + a[n] & \text{if } (n > 0) \end{cases}$$

إذا كانت المسألة الصغيرة نبرحصة أما المسألة بمسألة للفرصة فهي .

```
Float Rsum(float a[], int n)
{if (n<=0) return (0.0);
 Else return (Rsum(a,n-1)+a[n]);
}
```

ونفترض إن العنصره (a[1-3]=3 5 7) هي إلى (n=3)



بعضيات القر ع

نضع في فراغ مكان الدخول المتعاقبات الشكلية والمتخلفات العددية وعدوى للمعرف
كل تنسيق (استدعاء) يتطلب أربع حالات حرجية (حيزه بقيه (D) وحيزه للمعرف إلى العصفقوه
(E) وحيزه للمتعبر الذي يحفل اسم ندائه بالإصداغ إلى حيزه بطوان للمعرف
وهو في معنى السجل (عدد الاستدعاءات) نصب كالتالي.

معنى السجل (الاستدعاء) = الحجم الأولي في بول نشط - الحجم النهائي في آخر نشط
1 + |

معنى السجل (الاستدعاء) = $n + 1 + 4$

حدث في الحجم الأولي في أول نشط بعد به عدد العصفق n ثم نأخذ السجل في الحجم
النهائي في آخر نشط وهكذا إن يكون (D) أو في هذه أخرى.

$$T_{max}(n) = 4(n-1)$$

من المعقولة أن نلاحظ في الختي ندوة خطية بعد على قعدة (D) حدث إذا ارجاب قعدة
(E) أن لا الحرج أن إن قعد قعدة لا (D) كل للحرج

بعضيات أوقات مرفوعة بمتحارب علاقة السجل (Recurrence relations) بحسبها
كالتالي

$$f_{max}(n) = \begin{cases} 2 & \text{if } n \leq 0 \\ 2 + f_{max}(n-1) & \text{if } n > 0 \end{cases}$$

إن نأبعه (2) نأفل أرقن لفظلوب لاستدعاء كل تنفيذ ، لف (f_{max}(n-1)) فهي نأفل وقت
الندوة بقرن قر استدعاء
إن خطوة (Else) بعدر خطوة معالجة ندك فهي نأفل القعدة (D) وكن الخطوة التي
تحويباً نأفل نأبعه (I)
ولكن هذه العلاقة الداخلية تستخدم طريقة من طرق الحل وهي طريقة بقرن التكراري
(Iterative Substitution) كالتالي

$$\begin{aligned} f_{max}(n) &= 2 + f_{max}(n-1) \\ &= 2 + 2 + f_{max}(n-2) \\ &= 2(2) + f_{max}(n-2) \\ &= 2(2) + 2 + f_{max}(n-3) \\ &= 3(2) + f_{max}(n-3) \\ &= m(2) + f_{max}(n-m) \end{aligned}$$

و عندما $(m=n)$ تأتي

$$\begin{aligned} & 2n + t_{\text{main}}(n + n) \quad , \quad n > 0 \\ & = 2n + t_{\text{main}}(0) \\ & = 2n + 2 \end{aligned}$$

بعض المتغيرات في تكون ثابت $(n-n)$ قد تكون ثابت $(n-1)$ و أي قيمة تحتوي تحت في
تقوم إلى تكون ثابت $(n-m)$.

مثلاً : إذا كان مجموع عنصرين متتاليين كما في المتسلسلة n في

```

Cm+n = Am+n + Bm+n
Void Add(type a[][size], type b[][size], type c[ ][size] in m, m+n, n)
{ for(int i=1; i<=m; i++) ... m+1
  For (int j=1; j<=n; j++) ... Sum
    C[i][j]=a[i][j]+b[i][j]
}

```

نلاحظ في هذا المسألة يتصلب بالمتغيرات (m, n) حيث :

بعض المتغيرات : مثلاً مثلاً (m, n) نفس ذلك حزمة مقرر قيم المتغيرات (m, n, b, a)
بالإضافة إلى عنوان الذاكرة ونلاحظ أن الخوار لا يتوقف على خصائص المتغير في

$$S_{\text{main}}(m, n) = 0$$

في العادة إلى تحديد الوقت التي كالتالي

$$\begin{aligned} & m + 1 \\ & 5m \\ & (m + 1 + m) m \\ & (2m + 1) m \\ & 2mm + m + m + 1 \\ & T_{\text{main}}(m, m) = 2mm + 2m \end{aligned}$$

في هذه العلاقة تكون متعلقة في حصة $(m > n)$ أما عندما تكون $(m = n)$ فإنه يمكن حساب
تسهيبي لا (For) في ذلك حفظ تعقيد الذاكرة لتصبح :

$$T_{\text{main}}(m, m) = 2mm + 2m + 1$$

هناك أربع تعقيدات الزمن والوقت سلسلة عداد فيبوناتشي (Fibonacci) التي هي متتالية من الأعداد تبدأ بـ 0 و 1. أول حتمين قيع هو (0,1) وهما يندرج في المتتالية حيث أن عدله احصاءهم على الحبوب البياضيه من خلال جمع العددين السابقين كالتالي 0, 1, 1, 2, 3, 5, 8, 13.

جدد "n" لكل حد حدد من الحصول عليه من خلال جمع العددين السابقين به هذا كالم (F_n) يمثل الحد النوني في المتتالية قديم وبصورة عامة

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2}, n \geq 2 \end{aligned}$$

إن طريقة عمل أو تنفيذ البرنامج يتم بهندال عدد صحيح موجب ويتكلم (n) ويضيق قيمة (F_n) له حيث إننا كلفه (n=3) فإن (F_n=3) أو (n=4) فإن (F_n=6) إن جزء البرنامج المحسب بالمتابعة هو "

```
void Fibonacci (int n)
{ // Compute the nth Fibonacci number
  if (n<=1)                      +1
    Count<<n<<endl              +1
  else
  { int Fnum1=0,Fnum2=1,Fn      +-2
    for(int i=2;i<=n;i++)        n
    { Fn=Fnum1+Fnum2;            n-1
      Fnum1=Fnum2;               n-1
      Fnum2=Fn;                  n-1
    }
    Count<<Fn<<endl             +-1
  }
```

في حواصلنا هذا المثال لتعريف بالمتغير (n)

بعضات الخوارزم . نكتب البرنامج معه ثابت حركته بخبر أقدم المتغيرات (Fnum1, Fnum2, Fn, i, n) وعنوان للعودة وهو خزن ثابت ؟ بعدد على حواصلنا المثال أي إلى

$$S_{Fibonacci}(n) = 0$$

تعقيدات الزمن

بعد هـ عتبر حقائق التحليل تعقيدات الخوارزم بالبرامج وذلك لوجود شرط كالتالي . التحليل الأولي . عتد (n=0, n=1) في تعقيدات الزمن (عدد خطوات) هي (2). الحالة الثانية عتد (n=1) في عدد الخطوات هو (4n+1) وكما يلي .

$$T_{\text{Algorithm}}(n) = \begin{cases} 2 & \text{if } n \in (0,1) \\ 4n+1 & \text{if } n > 1 \end{cases}$$

ولنعم بعبارة حساب عدد الخطوات في الاجراء البرمجية التالية

```
int i = 1
while (i <= n)
{
  x += 1;
  i += 1;
}
```

نلاحظ ان الراس (while) يأخذ (n-1) من الخطوات مع مراعاة الانسيب إلى بداية الحداث المستخدم (i) والرعدة المضافة له.

بما في جزء (Do while) هذا نلاحظ ان (while) والخطوات الحثية به فعدد (n) من الحثيات

```
int i = 1
do {
  x += 1;
  i += 1;
} while (i <= n);
```

هنا : أوجد تعييرات التخزين و، توقف لتسأله حساب ما يعنى بالبرمطاف السبقة (Prefix Average) لتتابعه من الإعداد

يفكر برصيح التسأله كالآتي: بما كان يجب مصفوفة معينة ولكن (X) مخصصة بحري (n) من الإعداد مصفوفة فإن الخطوات حساب مصفوفة معينة هي (A) حيث ان نحصر (A1) نفس البرمط قيم الحاضر من (X[0] ... X[i]) لقيم (n-1, ..., 0) في أنه

$$A[i] = \frac{\sum_{j=0}^i x[j]}{i+1}$$

Algorithm Prefix Averages(x).

Input An n-element array x of number

Output An n-element array A of number

That A[i] is the Average of elements X[0], ..., X[i]

1.1 تحليل أفضل وأسى وأوسط حالة تنفيذ (Best & Worst & Average Cases Analysis,

يجب علينا ان نلاحظ هذا إذا كانت المسألة تأخذ بكثر من حالة وسوف نقوم بالتركيز على الحالة الاسوأ للعناصر لأنها تحوي تعقيدات كثيرة، كما يمكننا التخلص من الصعوبات في الحالات التي تكون فيها التعقيدات المعنوية (مضامين معنوية) غير مناسبة في كفاءة وحدها التحليل عدد الخطوات وذلك من خلال تحديد تلك أو مع من الخطوات.

أو عدد خطوات الحالة الافضل وهو التي عدد من الخطوات يمكن تنفيذها بفعالية عالية (أولى) عدد خطوات الحالة الاسوأ وهو أقصى عدد من الخطوات يمكن تنفيذها بفعالية معوية (ثاني) عدد خطوات الحالة المتوسطة وهو العدد المتوسط من الخطوات التي يمكن تنفيذها على أنه مسألة بفعالية معوية.

مثال // اوجد العنصر الأكبر في مصفوفة بعلاقة العدد

Algorithm Arranvmax (A,n)
Input An array A storing n integer
Output the maximum element in A

a[0] ← CurrentMax
1 to n-1 do ← For i
 If CurrentMax < A[i] then
 A[i] ← CurrentMax
 Endif
Endfor
Return CurrentMax

- الحالة الأفضل تكون البحث في أقصى حالاته عند تكون أول عنصر في المصفوفة هو الأكبر حيث لا يدخل في تنفيذ أي حلز (if)
- الحالة الاسوأ يكون البحث في أسوأ حالاته عندما يكون أخر عنصر في المصفوفة هو الأكبر حيث سيتم تنفيذ التقييم حتى الوصول إلى النهاية
- الحالة المتوسطة حيث نأخذ تعقيدات الحالة هي (عدد الخطوات لحد الوصول إلى العنصر (1,2,3, ... n) عدد الخطوات لكنه (n))

كذلك لاحظ ان مستوى التعقيدات يكون حسب الحالة ففي الحالة الأفضل تكون أقل تعقيدات وفي الحالة الاسوأ تكون أعلى تعقيدات.

ملحوظة // إلى طريقة حساب عدد الخطوات (Step Count) هي طريقة حساب تقريبية وليس دقيقة وهي صعبة نفس سرفا

$$T_{\text{Arithmetic}}^L(n) = 2n + 1$$

$$T_{\text{Arithmetic}}^R(n) = 3n$$

$$T_{\text{Arithmetic}}^A(n) = \frac{\sum_{i=1}^n (2n + i)}{n} = \frac{2n + \sum_{i=1}^n i}{n} = \frac{2n + \frac{n(n+1)}{2}}{n}$$

مع ملاحظة أنه في حاله المتوسطة فإنه يجري للعدد (i) إلى يبدأ من الصفر أو الواحد ، أما بد كلف عملية الحساب يبدأ بالعكس أي من نهاية إلى البداية فتكون

$$T_{\text{Arithmetic}}^A(n) = \frac{\sum_{i=1}^n (2n - i + 1)}{n}$$

5-1 الصيغ التقريبية (Asymptotic notation)

يوجد ثلاث صيغ تقريبية هي

- 1 صيغة الحد الأعلى (Big-Ob)
- 2 صيغة الحد الأدنى (Omega)
- 3 صيغة الحد الأوسط - الحد الأسى (Theta)

برهنت عدله تحدد عد الخطوات (Steps Count) على فني مهمة غشه في الصغره بذلك
لنحفظ إلى صيغ تقريبية لتحديد عد الخطوات

1 صيغة الحد الأعلى (Big-O)

ويقصد به إن تعقيدات الخزن أو الوقت ممكن أن تساوي بعد الأعلى أو تكون أقل منه و/أو
يمكن أن تكون أعلى منه وعندها صيغته تسمى كالأتي

$$f(n) \leq C \cdot g(n)$$

هذه المعادلة تطبق لنا فقط إن يوجد عدلان موحدان لها (C و n₀) بشرط أن

(C > 0 و n₀ > 1) حسب التقييد التالي

$$f(n) \leq Cg \quad n$$

نصنع لهم n ≥ n₀

نظرية: لا كلف

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n_1 + a_0$$

فلنعلم حدود درجتها (m) فلن -

$$f(n) = O(n^m)$$

وهذه بعض الأمثلة لتطبيق النظرية

مثال // ب كانت $3n + 2$ تمثل $T(n)$ (تقديرات خوارزمية) لأي برنامج معين في الحل ؟

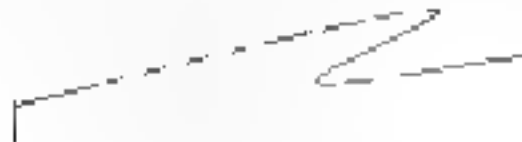
$$3n + 2 = O(n)$$

$$3n + 2 \leq 4n \quad \forall n \geq 2$$

بجميع قيم $n \geq 2$

حيث إن الكمية $(4n)$ تمثل التراتيب أي n هي ثابت $g(n)$ بمعنا 4 تمثل للثابت c ، حيث إننا نختار أكبر معامل 4 وهو (3) وأصغره واحد ليصبح (4) كافي علقه n $O(n)$ تمثل الحد الأعلى لتقديرات البرنامج .

وهذا يعني أنه علته أفضل التحصيل (النقاط الحقة) في منطقة معينة من دائرة مع التواء على شكل دائرة بدون تأثير كما في الشكل رقم (2) التالي



شكل (2) منطقة Notation O

- تعينه التصاعد منظمه
- على الأقل يوجد تعليه واحدة

ملاحظة // في الرموز مستخدم السانة تبدأ من الإحداثيات $(0,0)$ إلى $MaxX,MaxY$ لأن بعد الجزء المخطط فقط .

Example1 Use Big O Notation to analyze the time efficiency of following c++ code of the integer N

```
For(int i=1; i<=N/2; i++)
{
    For(int j=1; j<=N*N; j++)
    {
        // ...
    }
}
```

المعطى بين دوائر For() الخارجية مثلا (N,2) بينما For() الداخلية مثلا (N*N)
 $N/2 = N * N$
 $N/2 * N^2$
 $\frac{1}{2} N * N^2 = \text{big-O} = O(N^3)$

EX N=5;
 for(int k=1; k<=2; k++)
 for(int j=1; j<=25; j++)
 { }

Then $O(125)$

k	J
1	1
	.
2	25
1	1
	.
2	25

Example2 Use Big-O Notation to analyze the time efficiency of following c++ code of the integer n

```
For(int i=1; i<=n/2; i++)
{ }
For(int j=1; j<= n*n; j++)
{---}
```

$$n^2 + n^2 n = 1/2n + n^2 \quad n + n^1$$

$$n(1+n) \quad \text{Big-O} = O(n^2)$$

ملاحظة: لكسمة (1/2) و (1) يهملان ، على هذا كسمة واحدة يتركز لها (C) بوحدة (Big-O)
 لا ننتج إلى فرقت ، فلو فرقت فنتج

N=5,
 For(int i=1; i<=2; i++)
 For(int j=1; j<= n*n; j++)

هناك يحتاج (27) تكرار فقط للتكرار وهذا يعني إلى حالة السعيد هذا في أسوأ من الحالة السعيد
هناك لا تقرب من انه هناك المصعب التالي.

```

k=n
Do
{
  K=k/2
} While (k > 1);

```

هنا عندما تنقص قيمة n فهذا يعني انه الدالة $n \log n$ أو $\log n$ يتضاءل يمكن
أن يكون $O(n^2)$, $O(n)$ هي $O(n^2)$, $O(n^2)$

When $n=8$,
Then $k=8$,

k	الحقيقة
8	$8 > 1$
4	$4 > 1$
2	$2 > 1$
1	$1 \leq 1$

إن عدد مرات تكرار هذا المقطع ينقص إلى النصف في كل مرة هذا لأنه يتضاءل $\log n$
أي إلى (Big-O) له في $O(\log n)$ وسجده $O(n^2)$, $O(n)$, $O(n^2)$, $O(n^2)$ رده لها
تزيد من أضعه وحاصلها $n \log n$ لأنه تنصوب في n .

كما يمكن صياغة المثال بالصورة التالية
التيك للمعدة التالية

$3n + 2 \leq 4n$
المطلوب: إذا الصفحة التقريبية أي ثم إذا هذه (n) .

الحل: أي المعدة يكون شكله كالتالي:

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n_1 + a_0$$

بعد من المعدة فيها أول أو يساوي هذا يعني أن الصفحة في $O(n)$ (Big-O) وسجده على
 n هوها إلى على أضعه (n) هو الواحد وبالتالي يكون الحل الصفحة الأولى.

$$3n + 2 \leq 4n \quad O(n)$$

$$3n + 2 \leq 4n \quad \text{لا}$$

$$n \geq 2 \quad \text{بجميع قيم}$$

n	للطرف الأيسر	لتطرف الأيمن	$3n + 2 \leq 4n$	نتحقق
1	5	4	$4 \leq 5$	False
2	8	8	$8 \leq 8$	True
3	11	12	$12 \leq 11$	True
4	14	16	$16 \leq 14$	True

• الصيغة تحقق عندما قيمة الـ (n) أكبر من أو تساوي (2)

مثال 2: بنا كعب $10n^2 + 4n + 2$ يغش $P(n)$ (التعبئة حرجى ووقت) ليرتفع معنى
أوجد التقيد بدلالة الصيغ التقريبية .

$$10n^2 + 4n + 2 = O(n^2)$$

$$10n^2 + 4n + 2 \leq 11n^2 \quad \forall n \geq 6$$

كما يمكن حسابها السؤال بالصورة التالية
سواء نتحدث التالية

$$10n^2 + 4n + 2 \leq 11n^2$$

ف هي الصيغة المستطرفة رقيقة لـ (n)

الحل// بعد أن الصيغة أقل أو تساوي من الصيغة هي $(Big-O)$ و على أن (n) يغش أن
الـ (n) في الصيغة

$$10n^2 + 4n + 2 = O(n^2)$$

$$10n^2 + 4n + 2 \leq 11n^2$$

جميع قيم $n \geq 6$

التحقق	$10n^2 + 4n + 2 \leq 11n^2$	الصرف الأيمن	الطرف الأيسر	n
False	$1 \leq 16$	11	16	1
False	$44 \leq 90$	44	90	2
False	$99 \leq 104$	99	104	3
False	$176 \leq 178$	176	178	4
True	$275 \leq 272$	275	272	5
True	$396 \leq 286$	396	286	6
True	$847 \leq 786$	847	786	7

مثال 3: بنا كعب $(1 - 100)$ أوجد التقيد بدلالة الصيغ التقريبية ؟

الحل// بن قيمة 100 يغش $P(n)$ من الأيب (n) يمكن نقوله بالقيمة (1) ، فذا يعنى

$$100 \leq 100 + 1$$

جميع قيم $n \geq 1$

مثال 4: $5 + 2^n + n^2 = O(2^n)$ أوجد تقيد الحرجى ووقت بدلالة الصيغ التقريبية ؟

الحل// لاحظ أن هذا الترتيب هناك محاسب أسية و هي على محسوب ذلك إلى .

$$6 * 2^n + n^3 = O(2^n)$$

$$6 * 2^n + n^3 \leq 7 * 2^n \text{ لن } n \geq 4$$

نطبق قيم $n \geq 4$

ملاحظة: إذا وجدنا قيمة (n_0) بحيث تكون على n موجود في (n_0) السعة فهو لا يؤثر
ويعتبر أي سعة مختلفة.

$$\text{مثال 1/6: نحس من صحة المعادلة } 10n^2 + 4n + 2 \neq O(n)$$

نظن:

$$10n^2 + 4n + 2 \neq O(n)$$

$$10n^2 + 4n + 2 \leq 10^4 n$$

نطبق قيم $n \geq 10^4$ وهذا غير ممكن

$$\text{مثال 1/6: نحس من صحة المعادلة } 3n + 2 \neq O(1)$$

نظن:

$$3n + 2 \neq O(1)$$

$$3n + 2 \leq 10^4 + 1$$

نطبق قيم $n \geq 10^4$ وهذا غير ممكن

لأنه لا يمكن أن يكون $n \leq 10^4$.

2. صيغة أوميجا الكبرى (Ω)

ويقصد بها أن عقيدتي الحدود أو الوقت يمكن أن تكون أكبر من مستوى أوميجا الكبرى
ممكن أن تكون أقل منه في بعض الحالات يتم كتابتها كالتالي:

$$F(n) = \Omega(g(n))$$

بما وفقد إذا كان لدينا ثابت موجب (C, n_0) بحيث $(C > 0)$ و $(n_0 \in \mathbb{N})$ فإن:

$$F(n) \geq Cg(n)$$

نطبق قيم $n \geq n_0$

نظرية: إذا كانت

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$$

متعددة حدود من حيثها (n) فإن:

$$f(n) = \Omega(n^m)$$



شكل (3) بمسحه Notation Ω (يمكن)

- عملية التصاعد غير منقطعة (مستمرة)
- يوجد زاوية (0) وب قيمة
- ربما المعنى جيد عند 0 عند $[0, 2]$ وهناك استخدام أي دويقة خاصة سيحول أو يغير شكل الدالة تماماً

مثال: ليكن $3n + 2$ أقل من $3n$ ووقت البرهان معين أوجد هذه التطبيقات
بدلالة الصحيح القريبية ؟
الحل //

$$3n + 2 = \Omega n$$

$$3n + 2 \geq 3n \quad \forall$$

جميع قيم $n \geq 1$

لاحظ هنا مع n وهو (3) يعني كك هو أي إلى النأيب $c = 3$ ، كك يمكن بعين الصيغة
(1) $3n + 2 = \Omega n$ فهي لا تؤثر وتبقى الصيغة صحيحة

كك يمكن صياغة سؤال بالصورة التالية
نذكر المعتمد التالي

$$3n + 2 \geq 3n$$

المطلوب: إيجاد الصيغة القريبية ثم إيجاد قيمة n

$$3n + 2 = \Omega 3n$$

$$3n + 2 \geq 3n \quad \forall$$

جميع قيم $n \geq 1$

n	الطرف الأيسر	الطرف اليمين	$3n + 2 \geq 3n$	نتحقق
1	5	3	$5 \geq 3$	True
2	8	6	$8 \geq 6$	True
3	11	9	$11 \geq 9$	True

مثال 2/ // $\Omega(n) = 3n + 2$ ما هي المتعقدة ؟

نفس تعريف Ω السابقة هي Ω هذا يعني ان العلاقة هي \geq لتكون المتعقدة .

$$3n + 2 \geq 3n$$

مثال 3/ // نثبت لصيغة التقريبية التالية -

$$10n^2 + 4n + 2 = \Omega(n^2)$$

الخطوة الأولى تكون معرفة هذه الصيغة علماً ان $\{C=1\}$

تجرباً ، يف ان الصيغة هي الحد الأدنى هذا يعني ان الثابت يجب ان يكون أقل او مساوي للحد الأدنى اي 1

$$10n^2 + 4n + 2 \geq 10n^2$$

يصح قيم $n \geq 1$

مثال 4/ // نثبت صحة هذه المعادلة $\Omega(2^n) = 6 + 2^n + n^3$

الحل //

$$6 + 2^n + n^3 = \Omega(2^n)$$

$$6 + 2^n + n^3 \geq 6 + 2^n$$

لجميع قيم $n \geq 1$

3- صيغة الحد الأعلى Θ (Theta)

تستخدم الصيغة لتعريف حسب المتعقدة التي كتبت لها سابقاً

$$F(n) = \Theta(g(n))$$

إذا ربطت إذا وحدث تتوافق العوحد الثلاثة (C_1, C_2, C_3) بعدد يكون .

$$C_1 g(n) \leq F(n) \leq C_2 g(n)$$

لجميع لقيم $n \geq n_0$

نظريه إذا كانت

$$f(n) = a_n n^n + a_{n-1} n^{n-1} + \dots + a_1 n_1 + a_0$$

متعقدة حدود مرتبتها (n) هي .

$$F(n) = \Theta(n^n)$$



شكل (4): Notation: Θ (ثـ)

- كل تقريبه تغير الشكل تماما
- نفس الطول وكل مرة تحذف أو تضيف

مثال: أجب صحة العبارة $3n + 2 = \Theta(n^2)$ بالعبارة بدلالة صحة مقاربهه:
الحل: العبارة صحيحة

$$3n + 2 = \Theta(n^2)$$

$$\text{لأن } 3n \leq 3n + 2 \leq 4n$$

$$\text{بجميع قيم } n \geq 2$$

لاحظ هذا من تحديد صحة العبارة (2) يكون تصلي وليس عشوائي أي أنه (2) هذا فرق تحقق
الصيغة ليبدأ القيمة (1) لا تحقق بالصيغة

مثال: أجب العبارة التالية $3n \leq 3n + 2 \leq 4n$

الحل: لا فإنه يوجد غير على وهم مطلي وهذا يعني أنه يجب استخدام صيغة Θ

$$n \log n = \Theta(n \log n)$$

وبه في التواب (C) (C) أكبر من الصغر هذا يعني تحقق الشرط الأول سلك نقوم بتطبيق
الطريق

التحقق	الطرف الأيمن	الطرف الأيسر	الرمز	n
False	3	5	4	1
True	6	8	8	2
True	9	11	12	3
True	21	14	16	4

استنتج في الحل الصحيح أنه من $n \geq 2$

مثال 1/3: بين أن الصيغة التالية صحيحة

$$10n^2 + 4n + 2 = \Theta(n^2)$$

الحل: نريد أن نطابق قيمة Θ بـ n بصريا بالتوازي

$$10n^2 \leq 10n^2 + 4n + 2 \leq 11n^2$$

لجميع قيم $n \geq 5$

لاحظ أنه يجب أن نضع اعلی صفة للـ (n) في الحدین وبقعه الأقل للـ (C) نضع في الحید اليسری والأكبر في الجهة الیمنی

n	تصرف الأيسر	الأوسط	تصرف اليمين	التحقق
1	10	16	11	False
2	40	50	44	False
3	90	104	99	False
4	160	178	176	False
5	250	272	275	True

ملاحظة: في صيغة الحد Θ على الأی هو للصيغة الأكثر دقة والتحقق عندما يكون $\Theta(n)$ هو الحد الأعلى والأی للدالة $\Theta(n)$.

تمرین 1/4: برهن أن العلاقات التالية صحيحة

$$5n^2 - 6n = \Theta(n^2) \quad (C_1=5, C_2=11)$$

$$38n^2 + 4n^2 = \Omega(n^2) \quad (C=7)$$

تمرین 1/5: برهن أن العلاقات التالية غير صحيحة

$$10n^2 + 9 = \Theta(n)$$

$$n^2 + \log_{10} n = \Theta(n)$$

وهذه بعض حلول الأمثلة السابقة بطريقة الصحيح الخوارزمية.

$$S_{\text{avg}}(a, b, c) = \Theta(1)$$

$$T_{\text{avg}}(a, b, c) = \Theta(1)$$

بدأنا في صيغة الحد الأعلى تساوي صيغة الحد الأدنى فيما يمكن جيل استخدام صيغة التباين والعكس من ذلك وقد عد استخدام صيغة Θ (فقد يمكن استخدام صيغة Ω) أو Θ

$$S_{\text{avg}}(n) = \Theta(1)$$

$$T_{\text{avg}}(n) = \Theta(1)$$

$$\begin{aligned}
 S_{\text{avg}}(n, m) &= \Theta(1) \\
 T_{\text{avg}}(n, m) &= \Theta(m \cdot n) \\
 T_{\text{avg}}(n, n) &= \Theta(n^2) \quad \text{وفي حالة } m=n
 \end{aligned}$$

6-1 أوضاع الصيغ المتكررة لتقريب (The Times Of Recursive Notation)

يمكن توضيح الصيغ التي تستطيع من خلالها تحديد وفات التقدير بالمعادلة التالية
 $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

- إلى $O(1)$ تعني أنقذ من باقي الصيغ لأخرى وهكذا بالتمديد لبقية الصيغ هي إلى $O(\log n)$ في أفضل من باقي الصيغ التي بعدها
- الخوارزميات التي لها تعقيدات متزايدة أو وقت أكبر من $O(n \log n)$ بعد خوارزميات غير فعالة وكذلك فإن الخوارزميات التي تمتلك تعقيدات ضيقة أي $O(2^n)$ تكون ضيقة ولا تكون عملية إلا عندما تكون قيمة n صغيرة جداً أي أقل من 40
- ولتوضيح ذلك نقرر من يوجد حلف لنقطة 10^6 نقطة في الثانية إلى لحظة توقف تكون $P(n) = O(2^n)$

When $n=10$ then $f(n)=1$ ms
 When $n=20$ then $f(n)=1$ ms
 When $n=30$ then $f(n)=1$ sec
 When $n=40$ then $f(n)=18.3$ min
 When $n=50$ then $f(n)=1.3$ day
 4×10^{14} year When $n=100$ then $f(n)=$
 32×10^{14} year When $n=1000$ then $f(n)=$

تعتبر معطيات، أوجد حل مسألة في وقت أقل باستخدام أسلوب التداخل (الزمن عام 1997)

$$\text{Fibo}(n) = \begin{cases} n & n < 2 \\ \text{Fibo}(n-1) + \text{Fibo}(n-2) & \text{otherwise} \end{cases}$$

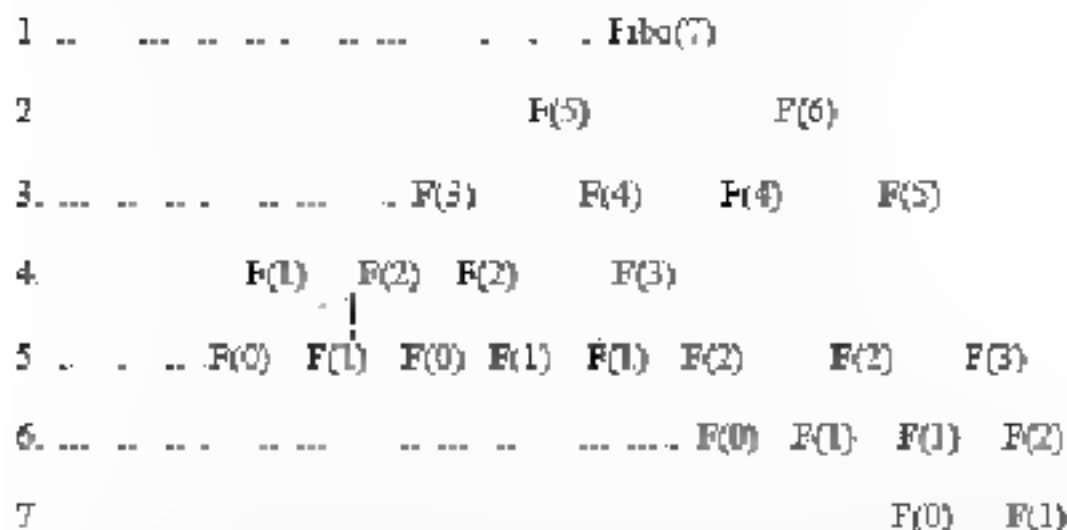
```

int Fibo1(int n)
{
    if (n < 2) return n;
    Else
        Return (Fibo1(n-1)+Fibo1(n-2));
}

```

بمسابك الوقت .

إن عملية حساب التكرارات بمسألة استدعاء ذاتي دائما يحتاج اليها إلى رسم مخطط تدويري يوضح تكرارات الاستدعاء . لذلك سوف نرسم شجرة تنشيطات بمسألة عددا فيبوناتشي كما في التالي:



1 - 2^k يمثل أقصى عدد للعقد في الشجرة . وبما أن كل عقد تمثل استدعاء فإن هذه القيمة تمثل عدد الاستدعاءات يقع داخلها هذه الشجرة التي حساب التكرارات للخاصة بكل استدعاء.

7.1 الاستدعاء في الشجرة التنشيطات (Recursion Tree).

ملاحظة: دعنا في الشجرة التكرارية أقصى عدد من العقد هو $(2^k - 1)$ حيث إن k يمثل عمق الشجرة (المستوى الأقصى) لذلك فإن تعقيد الوقت هو $O(2^k)$.

$$T_{Fib}(n) = O(2^n)$$

ونوضح ذلك:

عدد فيبوناتشي	عدد مرات تنشيط
7	1
6	1
5	2
4	3
3	5
2	8
1	13

نلاحظ أنه في مثالنا مثلا $Fib(30)$ فإن التنشيطات ستكون عددا ما يقرب 500,000 تنشط.

تعريف ٢٢: البحث الخطي هو الأسلوب المستخدم للعثور على العنصر المطلوب في مصفوفة الأرقام الخطية بدلاً من أسيه.

مثال: نكتب بعض الخوارزميات للبحث في المصفوفة الخطية باستخدام الأسلوب الخطي (Sequential Search) التي تتفحص في المصفوفة عن العنصر المطلوب في المصفوفة. البحث في المصفوفة الخطية هو الأسلوب المستخدم للعثور على العنصر المطلوب في المصفوفة الخطية. إذا كان العنصر موجوداً في المصفوفة، فإننا نرجع موقعه. وإلا، فإننا نرجع -1.

```
int SeqSearch ( Type a[] , Type x , int n)
{
    int i=0;
    while (a[i] != x)
        i++;
    return i;
}
```

الخط ٢٢: البحث في المصفوفة الخطية هو الأسلوب المستخدم للعثور على العنصر المطلوب في المصفوفة الخطية. إذا كان العنصر موجوداً في المصفوفة، فإننا نرجع موقعه. وإلا، فإننا نرجع -1.

١. حالة أفضل

$$T_{SeqSearch}^B(n) = \Theta(1)$$

٢. حالة أسوأ

$$T_{SeqSearch}^W(n) = \Theta(n)$$

٣. حالة متوسطة

$$\begin{aligned} T_{SeqSearch}^A(n) &= \frac{\sum_{i=1}^n (n-i+1)}{n} \\ &= \frac{(n+1)}{2} = \Theta(n) \\ &= \frac{1}{2} \cdot (n+1) = \frac{1}{2} \cdot n + \frac{1}{2} \end{aligned}$$

بما أن نسبة نصيب البحث في المصفوفة الخطية

$$T_{SeqSearch}^A(n) = \Theta(n)$$

فالمصفوفة الخطية هو الأسلوب المستخدم للعثور على العنصر المطلوب في المصفوفة الخطية. إذا كان العنصر موجوداً في المصفوفة، فإننا نرجع موقعه. وإلا، فإننا نرجع -1.

تعريفات المعية (Practical Complexities)

في تعريف الوقت لبرنامج معين يكون وبصورة عامة في دالة يخصص المثال وهذه الحالة مفيدة جداً في تحديد كيفية تغير متطلبات الوقت بتغير خصائص المثال باستخدام دالة التعقيد الصافي مع زيادة برنامج تقريبا مثل ما تحل نفس المسألة

ونفترض أنه لدينا البرنامج P يحوي تعقيداً زائفاً في $\Theta(n^p)$ وبرنامج Q يحوي

تعقيداً زائفاً في $\frac{Q}{\Theta(n^q)}$

نفساً في n أي البرنامج Q أفضل

ولكن هذا التعقيد Q علينا أن نتبع التالي

بأن n لكل برنامج جديد عندما يكون بعد أعلى والآخر يكون أقل هذا يعني أن

من بعد البرنامج P الذي حجمه n على هو OM بعده معقدة O ولجميع قيم

$n_1 \geq n$ حيث n تعقيد $O(n)$ و O تعقيد ثابت C

من بعد البرنامج Q الذي حجمه الإجمالي هو OM بعده معقدة O وبجميع قيم

$n \geq n_1$

وحيث $n_1 \leq OM \leq OM$ تعقيد $n \geq \frac{OM}{C}$

في البرنامج P يكون مربع من البرنامج Q تعقيد

$$n \geq \max \left\{ \frac{\alpha}{\beta}, n_1, n_2 \right\}$$

فإذا فرضنا أن البرنامج P يعقداً في 10^4 على فائدة بينما البرنامج Q يعقداً في n^2 معقداً يكون

$$n \leq 10^4$$

$$M^2 = \frac{t - b}{2t}$$

وفي حالة أنها أصبحت تربيعية فهذا يحصل قطع مكافئ

$$t = a_0 + a_1 n + a_2 n^2$$

أما إذا كانت الخطية فهي $O(nm)$ لأنه يحصل محطاً ذو صفحة

$$t = a_0 + a_1 n + a_2 n \log n$$

مثال: اقرص قبض، اجزئية سواء حلقة مغنولارية يجب التعقيد (Sequential Search)

المستخرج //

إلا أن الهدف من التجزئة هو قياس الحلقة الزمنية بحيث إن هذه سوارزمية تحتاج وقت قليل جداً وأقل وقت في الحذف هو أقل من جزء من الثانية لذلك يجب أن نعدّ دوائر برزانه الوقت كما في جزء البرنامج التالي

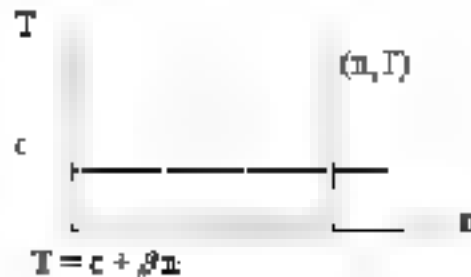
```
#include <iostream.h>
#include <iomanip.h>
#include <time.h>
void time Search()
{ // repetition factors
  Long int r[21]= {0,200,000,200,000,1,000,000, . . . 25000};
  Int a[1001],n[21];
  For (int j=1;j<=1000;j++) a[j]=j;
  For (int i=1;i<=10;i++)
  { n[i]=10*(i-1);
    n[i+10]=100*i;
  }
  Cout<<" n \t t "<<endl<<endl;
  Cout<<Setprecision(6);
  For (int j=1;j<=20;j++)
  { int h=Gettime();
    For (int i=1;i<=a[j];i++)
    { k=SeqSearch(a,0,n[j]);
      Int h1=Gettime();
      Int t1=h1-h;
      Float t =t1;
      cout<<" Setw(5)<<n[j]<<" Setw(5)<<t1<<" Setw(8)<<"<<endl;
    }
    cout<<" time are in millisecond <<endl;
  }
}
```

ويفيد بلى بوضوح نموذج التبريد.

- المصفوفة $[2][2]$ مستخدم لحزن قيم التكرارات التي تدور من 1 إلى 20 قيمة أي ألف موزن يُحدد 20 قيمة للـ n وهي غير خالية حيث إن القيمة من هذه التكرارات هي جعل الوقت n من الثانية مع ملاحظة أنه كلما قادت n من هذه التكرارات يقل ويقل. يجب تأكيد التكرارات
- إن شكل قيمة في مصفوفة الـ n تكرار يندرج في مصفوفة الـ r وحجم مصفوفة الـ n هو 2 حيث أننا نضعي في المصفوفة التي نبحث عنها والتي نعالق الموزن رقم 1 بالمصفوفة n بالمصفوفة g فهي تحوي العناصر التي نبحث فيها ليدها عن المصفوفة المطلوبة
- إن العلاقة التي نحصل من مصفوفة n نقدم هي معادلة حساب قيمة n ويمكن تقديرها من معادلات إلى تحري.
- إن الدالة $GetTime()$ هي جزء آخر مني يجد الوقت الحقيقي للتحسين n متسلسلاً بجزء n من القيمة هذا في القيمة 10 هذا يعني على القيمة 10 كانت القيمة 100 بقدره n قيمة
- المسطر n الذي خصصناه للتحقق من القيمة n المرفوعة ضمن المصفوفة الدالة يعني أي عنصر غير موجود بالمصفوفة n من القيمة يختلف عن قيمة n ويحوي الفرق في الوقت.
- الدالة $Setprecision(6)$ تحوي خاصية تظهر لنا بعد الفاصلة بالعدد العشري n بالـ $Set(5)$ فهي تحوي الانتقال منسلفه فحجم الرقم المخصص ضمن مسطر الحلق.

إن نتيجة هذا البرنامج هي الحصول على وقت الدالة (Sequential Search) مصنفاً إليه وهو حواف التكرارات والنظر في وقت دورة التكرارات على أن نجد نفس النتيجة بالوصول إلى حافة جسم التكرارات أي أننا نجعل جسم التكرارات فرعاً (for) ونسجل لنا في الـ n كل دورة ثم نطرحه من قيم n في التجربة السابقة (قيمة وقت الدورة للوحدة) فنخرج من كل دورة T هي التجربة السابقة كان وقت الدورة الواحد تقريباً (0.002) ولكنه غير ثابت في أنه قد يكون n من ذلك بكثير في التحسينات في تلك السرعة العالية

وهناك علاقة تربط حجم التكرارات (n) بالزمن (T) وهي علاقة الزمن بالحجم في الدالة Sequential Search



شكل (6) علاقة الزمن بالحجم في الدالة Sequential Search

الفصل الثاني

الترتيب

(Sorting)

2-3 خوارزميات الترتيب (Sorting Algorithms)

الخوارزميات هي عبارة عن مجموعة من الخطوات الحسابية و الرياضية و المنطقية المستخدمة لحل مشكلة ما ، و سنبين الخوارزميات بهذا الاسم نسبة إلى العالم الفيلسوف "أوجستر مكدون" وهو من الخوارزميين .

خوارزميات الترتيب هي خوارزميات يمكن من تقطيع مجموعة عناصر حسب ترتيب محدد العناصر الفريدة ترتيبها ، يوجد في مجموعة مرتبة بعلاقة ترتيب معينة

تصنيف خوارزميات الترتيب مهم جداً لأنه يمكن من اختيار نوع الخوارزمية الأكثر مناسبة للشكل المطلوب مع الأخذ بعين الاعتبار التقييم الموجودة في الخوارزمية

بمعنى آخر الترتيب عبارة عن عملية ترتيب مجموعة من العناصر التبادلية و التي قيمها معينة سمي حقاً أو ورق حقول تسمى مفتاح أو بصورة مصاعمية أو ترتيبية العناصر من الترتيب هو

1 - رتبة كتابة الخوارزمية " يجب عن عناصرها "

مثال // تمثيل الأرقام 3 و 4

3	4
النظام العشري	النظام العشري
11	100
النظام الثنائي	النظام الثنائي

0 0 0 0 1 1

0 0 0 0 1 0 0

استخدام 2 بيت للحرى

0 0 0 0 1 1

استخدام بيت 1

حسب في المساحة الحرة لأن لا صفر تسجل موقع

2 تبسيط معالجة الملفات

لأن الملفات تتألف من حقول من ترتيب هذه الملفات حسب مفتاح يكون تسجل في حصة الترتيب من البحث .

مثال // هناك بيت من تحت حقل هو بيت الترتيب و الثاني اسمه و الثالث العدد ، ستصبح أن نستخرج المعين للترتيب الأسماء حسب الترتيب و المعين

3 - حل مشكلة مشابه الفريد

يوجد مشكلة في الفريد هي مشابه الأسماء إذا كان الاسم مشابه لا يمكن أن يأخذ اسم الأب وإذا كان اسم مشابه فالحظ اسم الجد ، مثل اسم ريتاب

ريتاب حيدر محمد

ريتاب حيدر محمد

2-2 أنواع الترتيب (Types of Sorting)

- 1 الترتيب الداخلي (Internal Sort)
- 2 الترتيب الخارجي (External Sort)
- * الترتيب الداخلي يجب ان لا تكون حجم البيانات مناسب وليس كبير. ويشمل:
 - 1 ترتيب الاختيار (Selection Sort)
 - 2 ترتيب الفقاعي (Bubble Sort)
 - 3- ترتيب الإدخال (Insertion Sort)
 - 4- ترتيب قنبر (Shell Sort)
 - 5 الترتيب السريع (Quick Sort)
 - 6 ترتيب الأسس (Radix Sort)
 - 7- ترتيب المؤشرات (Pomiers Sort)
 - 8 الترتيب الشجري (شجرة البحث الثنائية) (Tree Sort)
 - 9 Topological sorting

* الترتيب الخارجي هو الترتيب الذي يجب خراج المعلومة الى أوسط الخزانة التقوي عند يكون حجم البيانات كبير بحيث يصعب استيعابها في الذاكرة. هذه عملية الترتيب ويشمل

- 1 الترتيب بالدمج (Merge Sort)
- 2 الترتيب بالدمج المتوازن أو المتساوي (Balanced Two Way Merge Sort)
- 3- الترتيب بالدمج باستخدام طريقة قسم وضم (Divided & Conquer Merge Sort)

تعتبر الترتيب المعقدة لا تفضل حواريه الترتيب .

- 1 حجم البيانات المطلوبة : كل صطل يكون حيث داخلي أو أما كان كبير يكون الحرج الخارجي .
- 2- نوع الخزانة : إذا كان دائرية وديسه يكون الحرج داخلي أما ² أنشودة مضطربة يكون الحرج خارجي.
- 3- درجة ترتيب البيانات : حيث إن ترتيب التجه مرقية لترتيب بشكل أسرع من الترتيب غير المتفرقة (مضطرب).

2 3 خوارزمية الترتيب الداخلي (Internal Sort):

1 خوارزمية الاختيار (Selection Algorithm)

ترتيب الاختيار هو خوارزمية لترتيب الأكثر بديهية ، و يتم عن طريق البحث إلى عن العنصر الأكثر أو عن العنصر الأصغر و سيقوم بوضع في المكان الأخير ثم يبحث عن ثاني أكبر أو أصغر عنصر و الذي يوضع في مكانه في حال العنصر الأخير ، سيقوم حتى يتم ترتيب الجدول بالكامل.

خصائص ترتيب جدول

- 1 عند اختيار رقم الصفه في ترتيب جدول عند عنصر V هو $2(N-1)$
- 2 عدد التباديل في ترتيبه N

ويمكن ان يصيح تلك خصائص الخصائص الآتيه

- 1 تحدد الصف في الصفه واستثنائه من موقعه مع العنصر في الموقع V في الصفه
- 2 تبعد الصف من العنصر في الصفه و استثنائه من موقعه مع الموقع الثاني في الصفه
- 3 تبعد الصف من الصفه حتى الوصول إلى العنصر في الموقع

مثال: ترتيب العنصر في الصف باستخدام طريقة الاختيار (Selection Algorithm)

8 9 7 6 4

Index	1	2	3	4	5	6
8	1	2	2	2	2	2
9	1	3	3	3	3	3
7	1	4	4	4	4	4
6	1	5	5	5	5	5
4	1	6	6	6	6	6
4	1	7	7	7	7	7

الاستنتاج

عدد العنصر $N=7$

عدد المراحل $N-1=6$

ملاحظة: معدل المقارنات هي $2(N-1) * N$ حسب

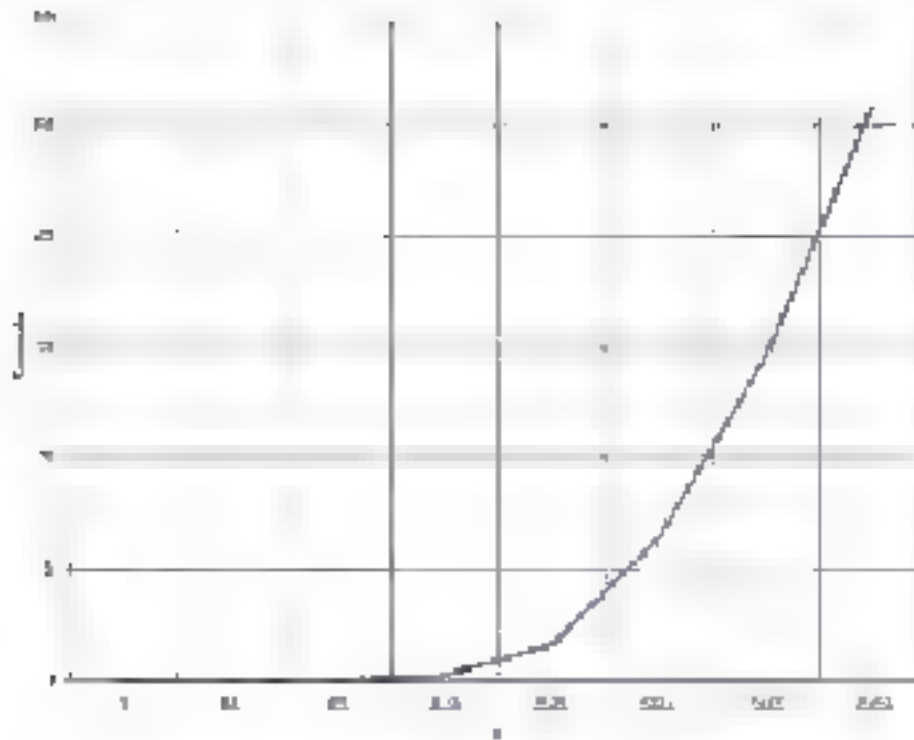
$$2 * 6 * 7 = 84$$

ملاحظة: عدد المقارنات بعدد على عدد المراحل ونساقص في كل مرحلة من المراحل إلى أن يصل إلى المرحله الأولى عدد المقارنات التي تكون بها هي $N-1=6$ ومن الملاحظات هو 21، ويمكن في طريقة الاختيار التي لاحظناها في هذه المرحله هي أن كل عنصر يمكن أن يكون في كل مرحلة من المراحل لأنه يمكن أن يكون في كل مرحلة من المراحل.

مثال: ترتيب صفه هي ثلاث عناصر مثل العنصر الذي هو (3) في الصف (2) ونصاح في الصف (1) ما العنصر (8) في الصف الثاني يعني في الصف الثالث وبنظر من تلك الملاحظة إلى مرحلة صفه أنه يجب أن يكون مع الرقم (9)

8	3	3
3	8	8
9	9	9

التحليل التجريبي (Empirical Analysis)



شكل (7) العلاقة بين حجم المدخلات (n) وكفاءة فرز الـ Selection Sort.

والدالة البرمجية التي تقوم بتطبيق هذه الطريقة هي

```
void selectionSort(int numbers[], int array_size)
{
    int i, j;
    int min, temp;
    for (i = 0; i < array_size - 1; i++)
    {
        min = i;
        for (j = i + 1; j < array_size; j++)
        {
            if (numbers[j] < numbers[min])
                min = j;
        }
        temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;
    }
}
```

هذه، دراجع يقوم باستخدام طريقة الترتيب بالاختيار لمجموعة كلمات

```
#include< stdio.h>
main( )
{ Char *z,
  Char *name[] {"ammar", "Fanna", "omar" "shameel", "jamal",
  "saeed", "yousef", "mariam"},
  Int nmax=8
  Register int i,j,k;
  For i=0; i<=nmax-1 ++i)
  { k=
    z=name[i],
    For(j=i+1, j<=nmax; ++j )
    {if (strcmp(name[j], z)< 0)
      { k=
        z=name[j],
      }
    }
    Name[k]=name[i];
    Name [i]=z,
  }
  For(i=0; i<=nmax; ++i)
    printf("%s<n" name[i])
}
Ahmed
Ammar
Fanna
jamal
Mariam
Omar
Saeed
yousef
```

2. خوارزمية الترتيب الفقاعي (Bubble Sort Algorithm)

ترتيب العناصر خوارزمية ترتيب بسيطة لطيفة، وهي تعمل على وضع العنصر الأكبر كفقاعة الهواء التي ترتفع إلى أعلى وذلك بترتيب العناصر بتتابع. أي نقوم بمقارنة العنصرين الأول والثاني، نحفظ العنصر الأكبر، و نبدل الأماكن إذا كانا غير مرتبين. نقوم بهذه العملية على كل عنصر، بعد ذلك نعيد ترتيب باقي العناصر من الأخير وهكذا نواصل حتى نصلح عدد وجود عنصر بالحد 1 في صفح لا نقوم بالتبديلات بعد بمر عليه

$$\frac{n(n-1)}{2}$$

لترتيب جثث A بعد n أفق عدد المقارنات سيكون:

$$\frac{n(n-1)}{2}$$

كما عدد التبادلات فهو في المتوسط

تقوم هذه الخوارزمية بترتيب عناصر n قسماً ثانياً على عدة مراحل عدد $n-1$ بحيث يتم وضع عدد واحد على الأقل في رتيبه الصحيح بنهايه كل مرحله

خطوات تطبيق الخوارزمية

- 1- إختيار الأعداد المراد ترتيبها في مصفوفة X
- 2- استخدام متحول $switched$ بقيمته على صورت ($switched=FALSE$) أو عدم حدوث تنبيل ($switched=TRUE$)
- 3- استخدام حلقه خارجية بعدد المراحل $n-1$ بحيث سوف الحلقة في حال عدم حدوث تبديل (أو عند هرقية)
- 4- استخدام حلقه متداخلة تقريية كل عدد بالعدد الذي يليه حيث يتم تغيير قيمة المتحول $switched$ إلى $true$ في حال التبديل
- 5- استخدام حلقه متداخلة لإظهار ترتيب الأعداد في نهاية كل مرحله
- 6- استخدام حلقه لإظهار ترتيب الأعداد النهائي

وهذا جزء البرنامج بالتطبيق الحصري

```
#include <iostream.h>
#define MAXNUM 10
enum boolean {FALSE, TRUE},
void main()
{int X[MAXNUM]
  int n, pass, hold;
  int switched=TRUE;
  cout<<"Enter count of numbers:"
  cin>>n;
  for(i=0; i<n; i++)
    cin>>X[i]
  for(pass=0; pass<n-1 && switched==TRUE; pass++)
  { switched=FALSE
    for(i=0; i<n-1; pass, i++)
      if(X[i]>X[i+1])
      { switched=TRUE;
        hold=X[i];
        X[i]=X[i+1];
        X[i+1]=hold;
      }
    }
```

```

for(i=0; i<n; i++)
    cout<<"X["<<i<<"]="<<endl;
    cout<<endl;
}
cout<<"The sort is: 'n ";
for(j=0; j<n; j++)
    cout<<"X["<<j<<"]="<<endl;
}

```

نذكره هذه الطريقة لتصنيف أيجاد أصغر العنصر الموجود في قائمة ما، ثم نبادله بالعنصر الأخير في القائمة.

- 1- First pass
- 2- Second pass

خطى عمل هذه الطريقة هي:

- 1- نأخذ العنصر الأول في القائمة ونضعه في المتغير N ، ونبدأ من العنصر الثاني في القائمة وننتقل إلى العنصر الأخير في القائمة ونأخذ العنصر الأصغر من العنصر N .
- 2- نأخذ العنصر الأصغر من العنصر N ونضعه في العنصر N .
- 3- نكرر الخطوات 1 و 2 من العنصر $N-1$ إلى العنصر الأول.

مثال: ترتيب العناصر في قائمة ما (Bubble Sort Algorithm).

8 3 9 7 2

List	pass(1)	pass (2)	pass (3)	pass (4)
8	8 3 9 2	2 2 2	2 2	2
3	3 3 2 9	8 8 3	3 3	3
9	9 2 3 3	3 3 8	8 7	7
7	2 9 9 9	7 7 7	7 8	8
2	7 7 7 7	9 9 9	9 9	9

عدد العناصر $N = 5$
 عدد التمريرات $N - 1 = 4$
 عدد المقارنات $N^2/2 = 25/2 = 12.5$
 عدد التبادلات $N^2/4 = 25/4 = 6.25$

من هذه الطريقة تكون جيدة إذا كانت العناصر متباعدة عن بعضها وعندئذ ينشأ كثير من المقارنات إلى مساحة تخزين كبيرة، بهذا فنحن نقف امام هذه الخوارزمية $O(N^2)$

3 خوارزمية الإدخال (Inserting Sort Algorithm)

نأخذ نفس هذه الخوارزمية كما يلي

1. نبدأ بالعنصر 2 في القائمة ونقارنه مع العنصر الأول ونضعه حيث الترتيب في مقادير القائمة وبذلك ترتيباً تصاعدياً.

2. نبدأ بالعنصر 3 ونقارنه مع العنصر الثاني ونضعه في المكان المناسب على العنصر الأول والثاني ونضعه في الموضع التالي ونسهر بالتسوية حتى الوصول على القائمة مرتبة

مثال: نأخذ العناصر الآتية بطريقة ترتيب الإدخال (Inserting Sort Algorithm)
8 3 9 7 2 6 4

الحل: يمكن ترتيبها كما في الجدول التالي

List	1	2	3	4	5	6
8	3	3	3	2	7	2
3	8	8	7	3	3	3
9	9	9	8	7	6	4
7	7	7	9	8	7	6
2	2	2	7	9	8	7
6	6	6	6	6	6	9
4	4	4	4	4	4	9

من الملاحظ الإدخال هو عكس ترتيب الاختيار لأنه يُنصّب العنصر ويقارنه مع العنصر الذي قبله حيث نأخذ الأول مع التالي والأول مع التالي وهكذا

عدد المقارنات $N=7$

عدد المراحل هو $N-1=6$

مجموع عدد المقارنات هو $N^2=4$

مجموع التبادلات هو $N^2=4$

عقل // البرنامج يقوم بتطبيق حوسبة مرتبة ترتيب الإدخال

```
typedef int tab_entiers[MAX];

void insertion(tab_entiers t) {
    /* Specifications externes */
    int i, p, x;
    for(i = 1; i < MAX; i++) {
        /* position d'insertion */
        /* determine p tel que p <= i */
        /* t[p] <= t[i] */
        p = 0;
        while(t[p] < t[i]) p++;
        x = t[i]; /* t[i] */
        for(j = i-1; j >= p; j--) t[j+1] = t[j];
        /* translation t[p+1] vers t[p+1+1] */
        t[p+1] = x; /* insertion t[i] */
    }
}
```

هذا البرنامج يصور مجموعة عناصر « العناصر المتتالية بالترتيب في العناصر المتتالية » أو « المتتالية » التي يراد ترتيبها. يتم العناصر المكتوبة بـ bold هي العناصر التي هي bold في العناصر العربية في مكانها الصحيح.

Initial array:

29	10	14	37	13
----	----	----	----	----

الحل : يمكن توضيحه بالخطوات التالية:

1.

29	10	14	13	37
----	----	----	----	----

2.

13	10	14	29	37
----	----	----	----	----

3.

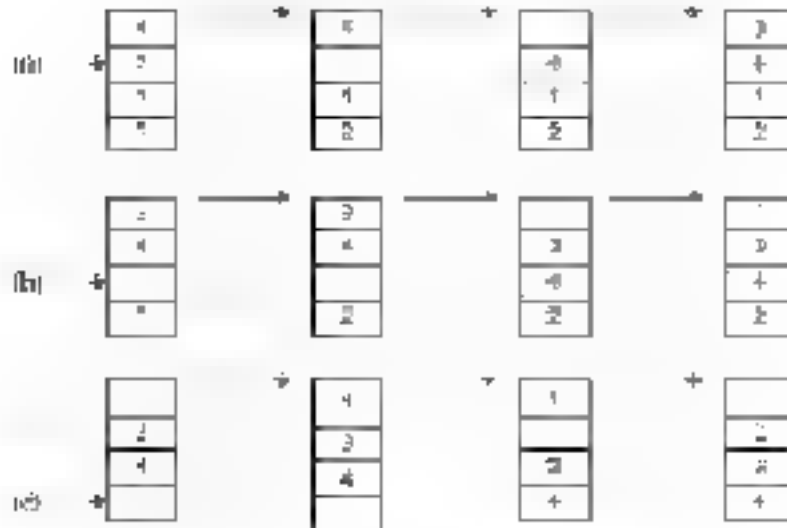
3	10	14	29	37
---	----	----	----	----

4.

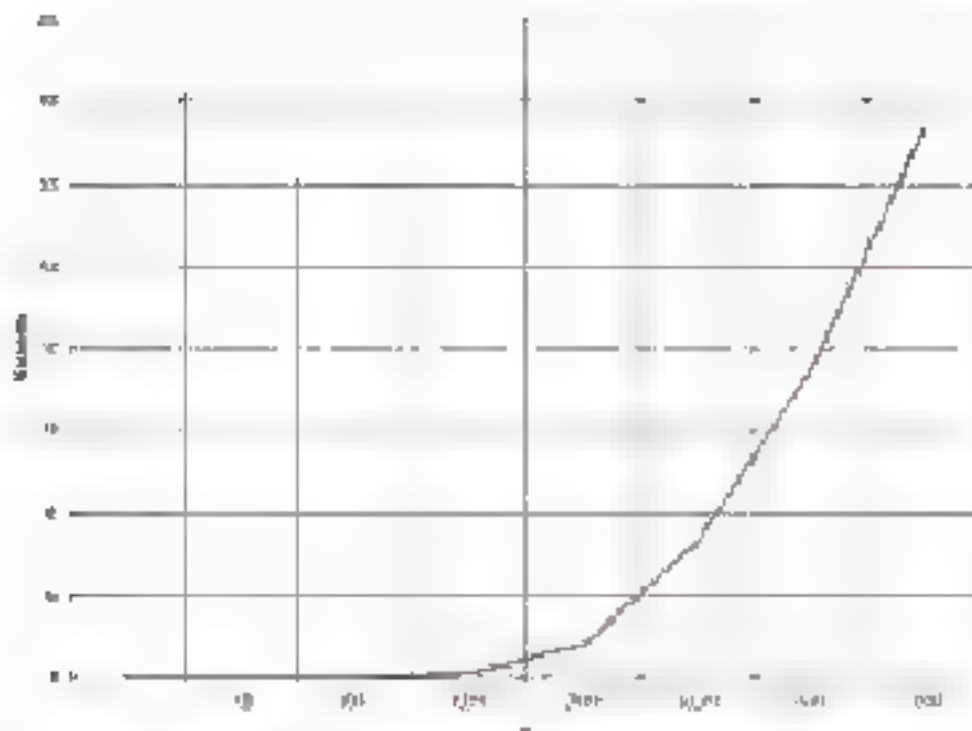
10	13	14	29	37
----	----	----	----	----

مثال: ديف عنصر الثامن، رتبة (4,3,1,2) باستخدام خوارزمية الإدراج.

الخطوة يمكن ترتيبها في الخطوات (a,b,c) التالية



تحليل تجريبي (Empirical Analysis)



شكل (9) كفاءة خوارزمية الإدراج (Insertion Sort Efficiency)

والحقبة البرمجية الخاصة بتطبيق خوارزمية الإدخال هي :

```
void insertionSort(int numbers[], int array_size)
{
    int i, j, index;

    for (i=1; i < array_size; i++)
    {
        index = numbers[i];
        j=i;
        while ((j > 0) && (numbers[j-1] > index))
        {
            numbers[j] = numbers[j-1]
            j = j-1;
        }
        numbers[j] = index;
    }
}
```

مثال // برنامج يقوم باستخدام خوارزمية بربيب لإزالة التكرار من مجموعة بيانات

```
#include<iostream.h>
using namespace std;
{
    Char *z;
    Char *name[ ]={'ammar','Fatima','omar','ahmed','jamel',
                  'saeed','youcef','mariam'};

    Int nmax=8;
    Register int i,j;
    For (i=0;i<=nmax-1; ++i){
        z=name[i],
        j= 1;
        While (j>=0 &&(strcmp(z,name[j])<0)){
            Name[j+1]=name[j]
            j
        }
        name[j+1]=z.
    }

    For (i=0, i<=nmax, ++i cout<<" " << name[i];
}
```

Ahmed
 Ammar
 Fatima
 jassal
 Mariam
 Omar
 Saeed
 yousef

3 خوارزمية شيل (Shell Sort Algorithm):

يوجد مشكلة في الخوارزمية الباعثي هي (في عدد العناصر بزيادة لكي، عنصر في زيادة عدد العناصر في الأعداد في الخوارزمية) هناك أن تكون العناصر في بعد ترتيب (في آخر عنصر في الخوارزمية) وأن الموضع الصحيح يجب أن يكون في الموضع الأول، هناك أنه في عدد كبير من العناصر وهذا يؤدي إلى كثير من الأخطاء.

ولذلك لابد من مشكلة من خلال خوارزمتين هي:

1- خوارزمية شيل

2 خوارزمية الترتيب السريع

لكنها تقلص كلاً من. حذف بعض المتقدم الخوارزمية إلى مسافة واحدة، وتحتوي على عنصرين أو أكثر ليس متجاورين وإنما على نفس المسافة لنفسه، ثم ينقسم المسافة الأولية إلى النصف ويحتوي العناصر أن تستمر ثلثه إلى أن تصبح المسافة تساوي 1، وبذلك يتم ترتيب الخوارزمية

في المسافة الأولية بين عنصرين تدعى فجوة (Gap):

مثلاً: هناك العناصر التالية 66, 5, 17, 93, 41, 11, 54, 43

الحل //

1 بحسب الأعداد المراد ترتيبها $N=8$

2 نضع المسافة الأولية في النصف 4 Gap



المرحلة الثالثة Gap 4/2 2

41 5 , 17 , 43 , 66 , 11 54 , 93



17 , 5 41 , 43 , 66 , 11 , 54 , 93



17 , 5 41 , 11 , 66 43 , 54 93



17 , 5 41 11 , 54 , 43 , 66 , 93

المرحلة الثالثة Gap 2/2 1

في هذه المرحلة تسهر عملية السحب حتى يحصل على القائمة مرتبة الإرتبة

5 11 17 41 , 43 , 54 , 66 , 93

خصائص تطبيق خوارزمية شل (Shell Sort Algorithm)

- 1 - يزيد كفاءتها كلما زادت عدد العنود
- 2 - لا بعد ج إلى مكان أصلي في القائمة لأجراء عملية الترتيب
- 3 - كفاءة إذا كانت القيمة داخل القائمة مرتبة أو شبه مرتبة
والسحب لكل خطوة على حدة
- 1 - يزيد كفاءتها لأنه يتم سحب العناصر في الأوصول التي {Gap}
- 2 - وذلك لأنه يمكن أن تكون مستقر أو غير مستقر الواحد سون استخدام مكان خاص له
- 3 - وذلك لأنه لا يوجد في هذ الخوارزمية بالارتداد كما كانت مرتبة أو عند الترتيب أقل من كفاءة الخوارزمية شبه مرتبة

تتميز خاصة تنظيم خوارزمية شل (Shell Sort Algorithm)

تستخدم هذه الخوارزمية للحصول على الحالة الأكثر ترتيباً في حالة الخوارزمية الكبيرة

تقدير الأول لتقدير معدل Gap حيث $Gap = 1.72^{\log(N^{1/3})}$

في المثال السابق استخدم

$$N = 8 \quad 4 \ 5 \ 6 \ 6 \ 6 \ 6 \ 6 \ 7 \approx 5$$

المعيار الثاني - تقسيم قيمة معدل الوقت (Time average) حيث -

$$T_{av} = \frac{N^2(5/3)}{8^{(2/3)} \cdot 32}$$

مثال: ارباب عناصر القائمة (3, 5, 1, 2, 4) بطريقة شبيهة باستخدام فجوة عدد (2) مرة واحدة (1) ؟

الحل: / يمكن ترتيبها كما في خطوات الآتية

(a) Gap=2

3	5	1	2	4
+	+	+	+	+

1, 2 3 5 4

نقسم الفجوة فكلون Gap=1

1	2	3	5	4
+	+	+	+	+

1, 2, 3, 4, 5

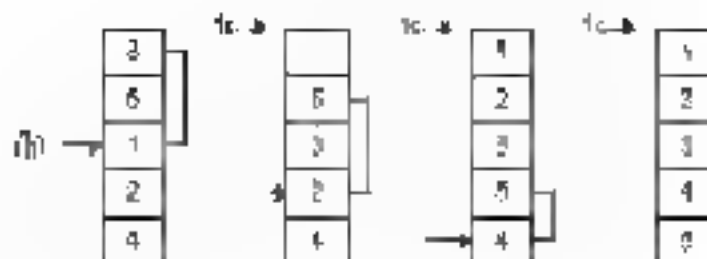
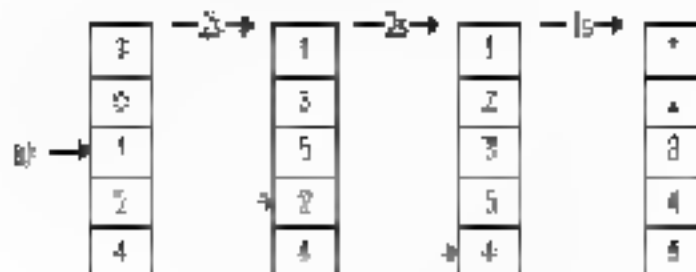
(b) Gap=1

3	5	1	2	4
+	+	+	+	+

نستقر بهذه الترتيب حتى فاصل على قائمة مرتبة

1, 2 3, 4, 5

ويمكن تمثيل هذه الترتيبات من عناصر القائمة بنفسه بالعصور الصغيرة كما يلي .

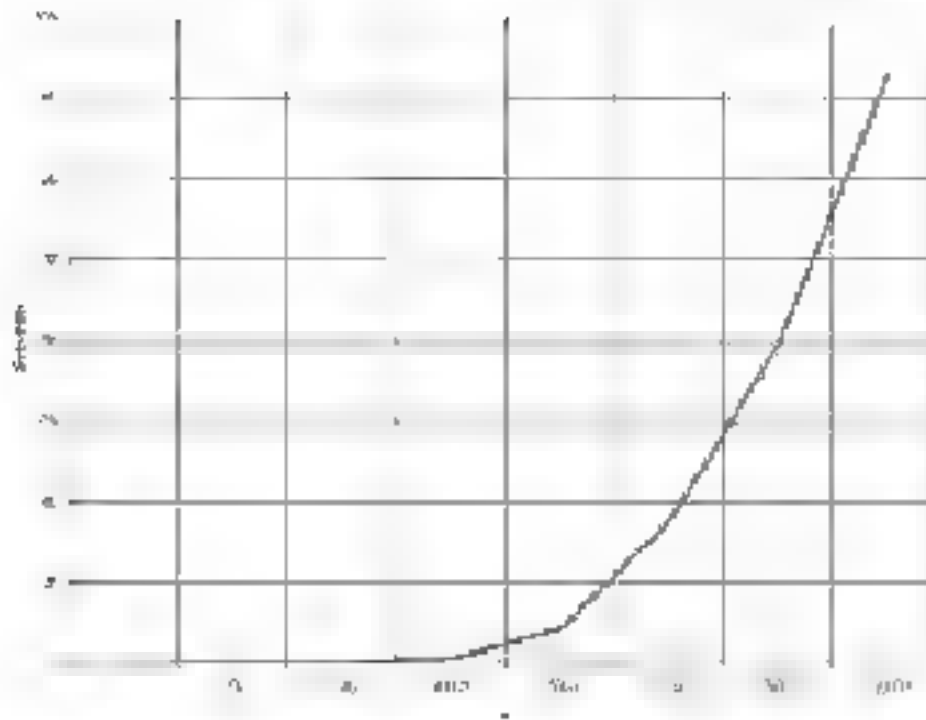


مثلاً: "أبي يفتح الباب" يستخدم خوارزمية ترتيب شياك (shell) ترتيب هجوعه كالات

```
#include< stdio.h>
main()
{
    Char *name[ ]={ 'anmar' "Fatima", "omar" "ahmed", 'jamaal'
                    "saeed" "yousef", "mariam"},
    Int m[ ]={9,5,3,2,1},
    Int nmax=8;
    Register int a,b,c,t,v;
    Char *z;
    For (v=0;v<=5; ++v){
        c=m[v];
        t=c;
        for (a=t; a<nmax; ++a){
            z=name[a];
            b=a-c;
        }
        If(b==0){
            t=t;
            t++;
            name[t]=z;
        }
        While((strcmp(z,name[b]))<0)&&(b>=0)&&(b<nmax){
            name[b+c]=name[b];
            b=b+c;
        }
        name[b+c]=z;
    }
    For(a=0; a<nmax; ++a)printf("%s\n", name[a],
}
```

Ahmed
Anmar
Fatima
jamaal
Mariam
Omar
Saeed
yousef

التحليل التجريبي (Empirical Analysis)



شكل (10): فعالية خوارزمية سيب (Shell Sort Efficiency)

والجزء البرمجي، نأخذ بسطين متساويين شياطين

```
void ShellSort(int numbers[], int array_size)
{
    int i, j, increment, temp;
    increment = 3; // increment = Gap
    while (increment > 0)
    {
        for (j = 0; j < array_size; j++)
        {
            i = j;
            temp = numbers[i];
            while ((i > increment) && (numbers[i - increment] > temp))
            {
                numbers[i] = numbers[i - increment];
                i = i - increment;
            }
            numbers[i] = temp;
        }
        if (increment > 1)
            increment = increment / 2;
    }
}
```

```

else if (increment == 1)
    increment = 0;
else
    increment = 1;
}
}

```

٥. خوارزمية الترتيب السريع (Quick Sort Algorithm)

الترتيب السريع هو طريقة ترتيب من اختراع هور (C A R Hoare) في 1962

خصائص خوارزمية

تتخذ الخوارزمية في عملها على وضع العنصر الأول (يسمى المؤشر) في مكانه النهائي ثم وضع العناصر الأكبر من المؤشر من جهة اليمين و العناصر الأصغر من جهة اليسار، وتسمى هذه العملية بمرحلة، ثم نقوم بحذف هذه المرحلة بمرحلة مقسمة لكل جهة (اليمين، اليسار) حيث نحدد مؤشر جديدا ونعيد عملية التجربة متكررة هذه العملية إلى أن نحصل على مجموعة مرتبة

إذا لم نحدد المؤشر بطريقة صحيحة، نحصل على الطريقة الأسرع للترتيب في لحظة المعلوماتية مع تعقيد $O(n \ln n)$ والتي قد تتحول إلى $O(n^2)$ في الحالة الأصعب، و في حالة جنوب عناصر مرتبة أصلا، و لكن هذه الحالة بسيطة لأن المجموعة مرتبة أصلا

من الناحية العملية، بالنسبة للتجربة مع عدد قليل لا يجاوز بضع عشرات من العناصر، يتم اللجوء عادة إلى الترتيب المباشر الذي يكون أفضل من الترتيب السريع

و بصورته عادة يعتبر الترتيب السريع الأكثر شيوعا (مبسطة) من بين جميع خوارزميات الترتيب حيث أنه مشكلة واضحة يمكن في كافة الحواسيب المؤشر.

اختيار أفضل مؤشر

بعد استعمال الترتيب السريع مجموعة مرتبة مسبقا، و طريقة عملها، نستقرى كيف قلت وقت كثيرا، و ذلك بسبب أن نزل عنصر هو الذي يعتبر هو من الأشياء التي يؤدي إلى عدم استخدام المجموعة التي هي أكبر، و استقرى من المؤشر. لحل المشكلة يتم اختيار عنصر الوسط، كما يمكن اختياره عشوائيا من عنصرين متواجدين حول المركز

تكون فكرة خوارزمية الترتيب باستخدام مبدأ التجربة حيث نقوم بعمل الخطوات التالية:

- 1- نقسم القائمة إلى جزئين حيث نأخذ عنصر للقسمه و نأخذ في الوسط تقريبا بعينه (X)
- 2- نقوم بعينه لتصبح بالحددين بحيث تكون العناصر على جهة اليسار هي الأصغر من (X) و هنا نقوم بتقسيم، و العناصر الموجودة على جهة اليمين فهي الأكبر من قيمة (X)

3. نأخذ النصف الأول ونجري عليه عملية مشابهة لذلك سريع مرة أخرى كذلك نأخذ النصف الثاني وهكذا إلى أن تكون جميع العناصر مرتبة

(النصف الثاني (اليمين والأكبر) (X) (النصف الثاني (اليسار و الأصغر)

ملاحظة

* إذا كانت قيمة (N) هي عدد زوجي فإن قيمة (X) تكون

مثال N=8

$$8 \div 2 = 4$$

نكتبه كـ السطر : 1 2 3 4 5 6 7 8

* إذا كانت قيمة (N) هي عدد فردي فإن قيمة (X) تكون

مثال N=11

$$11 \div 2 = 5.5$$

5 or 6

نكتبه كـ في الشكل : 1 2 3 4 5 6 7 8 9 10 11

مثال 4. رتب العناصر التالية ترتيب تصاعدي باستخدام الترتيب السريع (Quick Sort Algorithm)

20 , 85 , 60 , 75 , 70 , 88 , 50 , 90 , 33 , 95

الحل // نعمل باستخدام الصغير التالي

$$X = 10/2 = 5$$

المعرف X : العنصر الموجود في وسط القائمة
القيمة X = 70

F = Front : متعة القائمة ونكتب بالحد ()

L = Last : المؤخرة القائمة ونكتب بالحد ()

المرحلة الأولى X = 5

20 , 85 , 60 , 75 , 70 , 88 , 50 , 90 , 33 , 95

$$I = 1 \quad F = 1$$

$$J = 10 \quad L = 9$$

20 < 95 أي لا يوجد تبديل

20 85 60 75 70 88 50 90 33 95

$$I = 2 \quad F = 2$$

$$J = 9 \quad L = 9$$

85 < 33 أي يوجد تبديل

20 33 60 75 70 88 50 90 85 95

$I=3 \quad F=3$
 $I=8 \quad L=8$
 $60 < 90$ أي لا يوجد تبادل
 $20, 33, 60, 75, 70, 88, 90, 85, 95$

$I=5 \quad F=5$
 $I=6 \quad L=6$
 $70 < 88$ أي لا يوجد تبادل

$20, 33, 60, 90, 70, 88, 75, 90, 85, 95$

$I=4 \quad F=4$
 $I=7 \quad L=7$
 $50 < 75$ أي يوجد تبادل

$20, 33, 60, 90, 70, 88, 75, 90, 85, 95$
 $I=5 \quad F=5$
 $20, 33, 60, 90, 70, 88, 75, 90, 85, 95$

المرحلة الثانية-

$20, 33, 60, 50, 70, 88, 75, 90, 85, 95$
 $I=1 \quad J=4 \quad I=6 \quad J=10$

$X = 33$
 $70, 33, 60, 50$
 $\leftarrow X \rightarrow$

$X = 90$
 $88, 75, 90, 85, 95$
 $\leftarrow X \rightarrow$

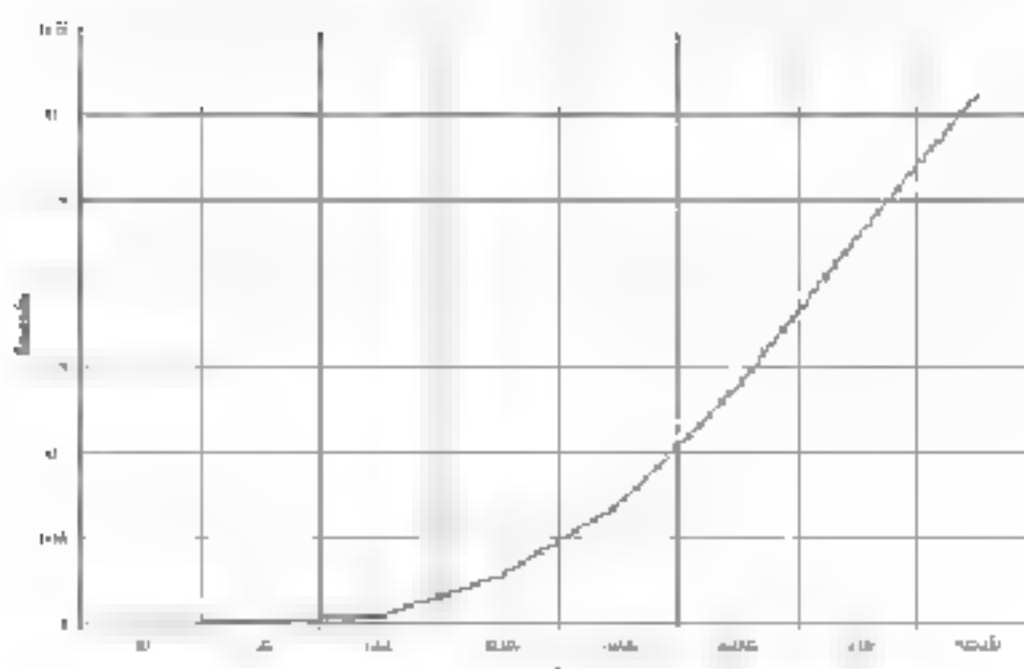
ملاحظة// في خوارزمية الفرز القسري السريع تمتعق الكثير من الوقت خاصة ان كنا نعد
 العناصر كغير حيث أن
 $N \log_2 N$ معدل معدل التعقيد
 $N \log_2 N$ معدل التعقيد

مثال ١٠: رتب عناصر القائمة الآتية (١, ٢, ٣, ٤) باستخدام خوارزمية فرز سريع ^{*}

الحل // يمكن ذلك كما في الخطوات الآتية



تحليل التجريبي (Empirical Analysis)



شكل (11) كفاءة خوارزمية فرز سريع (Quick Sort Efficiency)

والجزء اليماني الخاص بتطبيق خوارزمته اقرب الى المربع هو

```
void quickSort(int numbers[], int array_size)
{
    q_sort(numbers, 0, array_size - 1)
}
void q_sort(int numbers[], int left, int right)
{
    int pivot, l_hold, r_hold;

    l_hold = left;
    r_hold = right;
    pivot = numbers[left];
    while (left < right)
    {
        while ((numbers[right] >= pivot) && (left < right))
            right--;
        if (left != right)
        {
            numbers[left] = numbers[right];
            left++;
        }
        while ((numbers[left] <= pivot) && (left < right))
            left++;
        if (left != right)
        {
            numbers[right] = numbers[left];
            right--;
        }
    }
    numbers[left] = pivot;
    pivot = left;
    left = l_hold;
    right = r_hold;
    if (left < pivot)
        q_sort(numbers, left, pivot - 1);
    if (right > pivot)
        q_sort(numbers, pivot + 1, right)
}
```

تجربيات أخرى

عند استعمال الترتيب السريع لترتيب هجرو عنه تأكد عناصر كثيرة يمكن تغيير تعينه
التي تلك عدد الوصول إلى عضو عن جرتة غير مرتبة عدد عناصرها صغير أي 10 عناصر أي
أكثر بقليل الترتيب بالاختيار مناسب في هذه الحالة

مثال / استخدام برآل تقوم بتقسيم شجرة كثيرة إلى عناصر جرتية أصغر ورشيح باستخدام
حوالته للترتيب السريع

```
typedef int tab_entiers[MAX];
```

```
int rapideEtape(tab_entiers t, int min, int max) {
    int temp = t[max];
    while (max > min) {
        while (max > min && t[min] <= temp) min++;
        if (max > min) {
            t[max] = t[min];
            max--;
            while (max > min && t[max] >= temp) max--;
            if (max > min) {
                t[min] = t[max];
                min++;
            }
        }
    }
    t[max] = temp;
    return max;
}
```

```
void rapide(tab_entiers t, int deb, int fin) {
    int mil;
    if (deb < fin) {
        mil = rapideEtape(t, deb, fin);
        if (mil < deb > fin - mil) {
            rapide(t, mil + 1, fin);
            rapide(t, deb, mil - 1);
        }
        else {
            rapide(t, deb, mil - 1);
            rapide(t, mil + 1, fin);
        }
    }
}
```

مثلاً برنامج يوضح كيفية امتداد النوار التي تقوم بعملية التقسيم السريع
(Quick Sort Algorithm)

```
Sort(A)
  Quicksort(A,1,n)

Quicksort(A, low, high)
  if ( low < high)
    pivot location = Partition(A,low,high)
    Quicksort(A,low, pivot location - 1)
    Quicksort(A, pivot location+1, high)

Partition(A, low, high)
  pivot = A[low]
  leftwall = low
  for i = low+1 to high
    if (A[i] < pivot) then
      leftwall = leftwall + 1
      swap A[i] A[leftwall]
  swap(A[low] A[leftwall])
```

الجدول التالي يوضح وقتاً بالثواني لخمسة خوارزميات (المتوسط، أفضل، وأقرب السريع) =

method	statement	average time	worst-case time
insertion sort	9	$O(n^2)$	$O(n^2)$
shell sort	17	$O(n^{1.25})$	$O(n^{2.5})$
quicksort	21	$O(n \log n)$	$O(n^2)$

count	insertion	shell	quicksort
16	39 μ s	46 μ s	51 μ s
256	4,968 μ s	1,230 μ s	911 μ s
4,096	1.315 sec	.033 sec	.000 sec
65,536	436.427 sec	1.254 sec	461 sec

6. خوارزمية الترتيب الأسفل (ترقيعي) (Radix sort Algorithm)

سرع من الترتيب يعتمد على السرعة الموجودة في الارقام وتنقسم إلى حلقاء بحيث ترتب العناصر حسب الارقام (Pockets) على هذه الطريقة يستخدم ما يسمى الخلفاء (Digit) (0-9) بعدد الحلقاء في كل مرحلة بحيث تكون عدد الارقام مساوي في أكبر عدد الارقام في أكبر رقم.

مثال تطبيق // إذا كان لدينا العنصر التاليه (9 7 132)، فإن أكبر رقم يحتوي على 3 مراتب هو (132)

ملاحظة: من هذه الطريقة تعتبر غير عملية وذلك لإعداد استخدام الذاكرة الاحتياط في كل مرة بحيث تعتبر (link Queue)، أي أن كل حلقه هي بمثابة طابور عناصر FIFO

مثال: هناك العناصر التالية وبها نستخدم لترتيب ترقيعي (Radix sort Algorithm).

42	23	74	11	65	58	94	36	99	87
Ordinal									

الحل: يمكن توصيحه بجداول لكل حلقه (مرتبعة) وبها يكون:

0	1	2	3	4	5	6	7	8	9
	11	42	23	74, 94	65	36	87	58	99

1-Pockets1 11,42,23 74,94,65,36,87,58,99

0	1	2	3	4	5	6	7	8	9
	11	23	36	42	58	65	74	87	94 99

2- Pockets2 11,23,36 42,58,65, 74,87 94 99

7 ترتيب المؤشرات (Sorting Pointers Algorithm)

سنستخدم هذه الطريقة لربط وترتيب العناصر حسب المؤشرات حيث سنستخدم فكرة التحويل كما هي الحال التالي .

مثلاً: نأخذ العناصر الآتية بطريقة ترتيب المؤشرات (Sorting Pointers Algorithm)

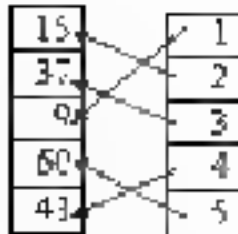
15 , 37 , 9 60 , 43

الخط ١: يمكن ترتيبها بالطريقة الآتية .

- ١ - صنع العناصر لي خائف
- 2 - صنع المؤشرات

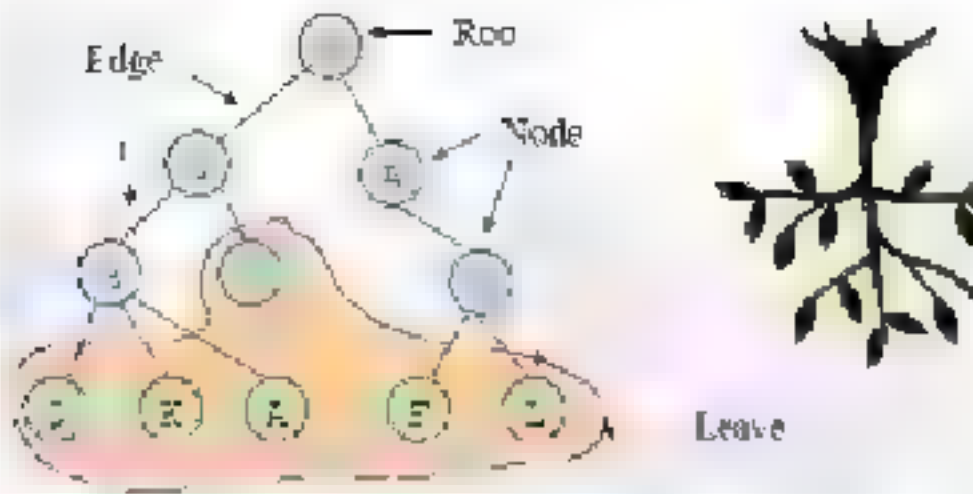


ب ترتيب المؤشرات



8 الترتيب بشجري شجرة بحث الثنائية (Tree for Binary search tree)

الشجرة هي بنية تكون فيها القيمة الأولية لأي عدة أكبر من البنية الأيسر به وبالعكس من القيمة التي هي الأيمن لها. يسمّى من الجذر (Root) والعقد (Nodes) والأوراق (Leaves) وهي خطوط عمقه وكذلك سطح خطوطه. معناه أيضاً ، والشكل رقم (12) الآتي يوضح الهيكل الشجري.



شكل رقم (2) الترتيب الشجري

مثال تطبيق: شجرة بحث ثنائية، يتم بناءها من حساب الأعداد التالية الشجرى



مثال: كود شجرة بحث ثنائية (Tree Binary Search) العناصر التالية بعد كل خطوة ؟
5, 9, 7, 3, 8, 12, 6, 4, 20

الحل: يمكن ذلك من خلال الخطوات الآتية:

1-

نحدد العنصر الأول فهو 5

2-

نحدد العنصر الثاني فهو 9، جوع 9 من 5 لأنه أكبر من 5

3-

(9)

3. نأخذ العنصر الثالث فنكون فرعاً من العنصر الثاني



4. نأخذ العنصر الرابع فنكون فرعاً من العنصر



5. نأخذ العنصر الخامس فنكون فرعاً من (7) (العنصر الثالث)



6. نأخذ العنصر السادس فنكون فرعاً من (5) (العنصر الثاني)



7 - نأخذ العنصر السابع (6) فنكون فرعاً من (7) (العنصر الثالث)



8. العنصر الثامن (4) نكرر فرعاً من (3) (العنصر الرابع)



9. العنصر بالتسليم 20 يكون فرع يفرع منه (2, 1)



3 4 6 7, 8 9 12 20

بما اننا نستخدم الحذف الخاصة بالاشجار الثنائية فإنه يمكن توضيحها بالطرق الثلاث
الآتية:

1- حذف عقدة نهائية (ورقة) - (Leave Delete):

لكي نقوم بحذف عقدة نهائية فائماً يجب ان نغير العقدة وننوي أن نؤشر على عقدة أخرى



قبل الحذف



بعد الحذف

2. حذف عقدة لها من واحد (One Child Node Delete):

لكي نقوم بحذف العقدة هذه نطبق ما يلي:

1- جعل المؤشر في العقدة يشير إلى العقدة الأخرى

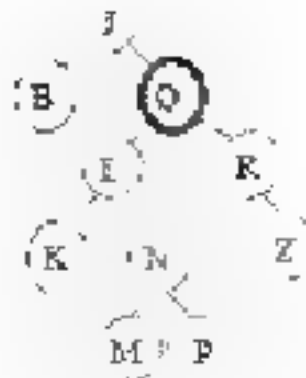
١-ا- نفي العقد المتصور dispose



١-ب- حذف عقده بعد فرز (Two Childs Node Delete)

- ١- يتم ذلك من خلال الخطوات التالية
- ٢- استفس العقد المطلوب حذفه، بالعثور على العقد الذي له واحد من العقدتين
- ٣- الفرعية اليسرى أو الشجرة الفرعية اليسرى بالعبء للعقد
- ٤- نسخة الشجرة الفرعية اليسرى للعقد التي للعقد في السابق للعقد المطلوب حذفها ، علف فئة أنه
- ٥- بديل لها فرع اليسر ليكون الفرع اليمين بديل

الشجرة التالية توضح عملية حذف العقد Q واستبدالها بالعقد P





2 حذف النقطة (4) وهي عقدة نهاية (ورقة)



3 حذف النقطة (5) وهي عقدة الجذر وتعاين في 6 نفس فقط لنا في الإزاحة إليها



4- حذف النقطة (6) عقدة نهاية



5 حذف عقدة (7) ، حيث في نفس فقط لنا فهو في 8



6 حذف عقدة (3) عقدة نهاية



7- حذف عقدة (9) . تلك رقم ليس لها نود يربتها



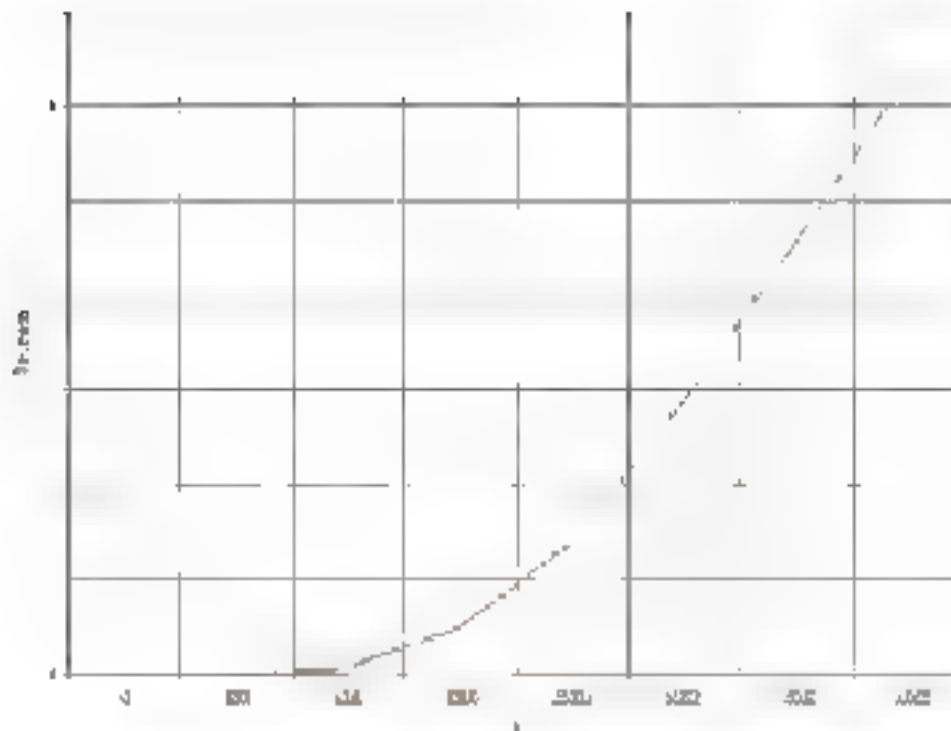
8- حذف عقدة (12) . تلك رقم ليس لها نود يربتها



و نوضح هذه النود يربتها داخل الميكس كالتالي

3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	----	----

تحليل تجريبي (Empirical Analysis).



شكل (13): فعالية خوارزمية الترتيب التجريبي (Tree Sort Efficiency)

و جزء البرنامج الذي يقوم بعملية الترتيب للشجرة هو

```
void TreeSort(int numbers[], int array_size)
{
    int i, temp;
    for (i = (array_size / 2) - 1; i >= 0; i--)
        siftDown(numbers, i, array_size);
    for (i = array_size - 1; i >= 1; i--)
    {
        temp = numbers[0];
        numbers[0] = numbers[i];
        numbers[i] = temp;
        siftDown(numbers, 0, i - 1);
    }
}

void siftDown(int numbers[], int root, int bottom)
{
    int done, maxChild, temp;
    done = 0;
    while ((root * 2 <= bottom) && (!done))
    {
        if (root * 2 == bottom)
            maxChild = root * 2;
        else if (numbers[root * 2] > numbers[root * 2 + 1])
            maxChild = root * 2;
        else
            maxChild = root * 2 + 1;

        if (numbers[root] < numbers[maxChild])
        {
            temp = numbers[root];
            numbers[root] = numbers[maxChild];
            numbers[maxChild] = temp;
            root = maxChild;
        }
        else
            done = 1;
    }
}
```


٩. خوارزمية الترتيب التوبولوجي (Topological sorting Algorithm)

تستخدم خوارزمية الترتيب هذه فكرة للمصنفات ووضح التيم يانطير، ويمكن توضيحها بالمثل التالي



The graph shown to the left has many valid topological sorts, including

- 1,5,3,11,8,2,10,9
- 7,5,11,2,3,10,8,9
- 3,7,8,5,11,10,9,2
- 2,5,7,11,10,3,8,9

من المصنفات الطوبولوجية هي: وقت التنفيذ خطي، يعتمد بريفيد العقد بين المسارات المختلفة، وقد استخدمت إحدى الخوارزميات كانت من قبل العالم (Kahn 1962)، معتمدة لفكرة الطوبولوجيا

حيث: E

E = الحواف

Q = الطوبولوجيا

V = الرؤوس

والخوارزمية التي توضح الطريقة هي

```

L ← Empty list where we put the sorted elements
Q ← Set of all nodes with no incoming edges
while Q is non-empty do
  remove a node n from Q
  insert n into L
  for each node m with an edge e from n to m do
    remove edge e from the graph
    if m has no other incoming edges then
      insert m into Q
if graph has edges then
  output error message (graph has a cycle)
  
```

else

output message (proposed topologically sorted order 1)

2-4. في رسم الترتيب الحد هي (External Sorting Algorithms).

1. خوارزمية ترتيب الدمج (Merge sort Algorithm).

هذه، تقوم بترتيب مجموعة عناصر بطريقة الدمج، كما في الجدول التالية

مرحلة	T1	T2	T3
A	7 8 8 4	0 9	
B	6		9 8 7 4
C	0 8 7 4	2 6	
D			9 8 7 6 5 4

وجاء الترتيب الخاص بتطبيق طريقة ترتيب الدمج هو

```
void MergeSort(int numbers[], int temp[], int array_size)
{
    m_sort(numbers, temp, 0, array_size - 1);
}
void m_sort(int numbers[], int temp[], int left, int right)
{
    int mid;
    if (right > left)
    {
        mid = (right + left) / 2;
        m_sort(numbers, temp, left, mid);
        m_sort(numbers, temp, mid+1, right);
        merge(numbers, temp, left, mid+1, right);
    }
}
```

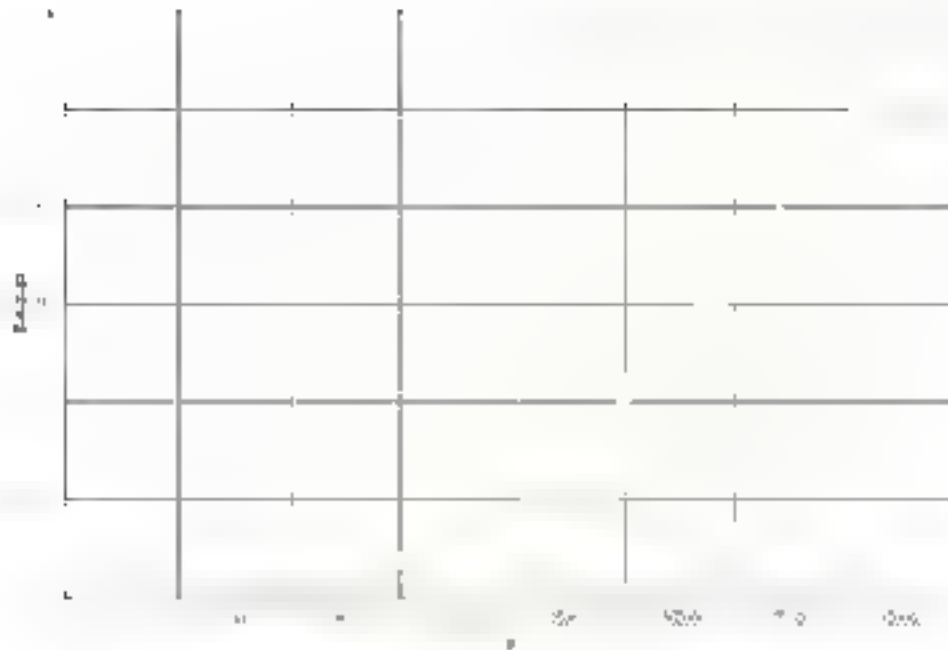
```

}
void merge(int numbers[], int temp[], int left, int mid, int right)
{
    int i, left_end, num_elements, tmp_pos;
    left_end = mid - 1;
    tmp_pos = left;
    num_elements = right - left + 1;

    while ((left <= left_end) && (mid <= right))
    {
        if (numbers[left] <= numbers[mid])
        {
            temp[tmp_pos] = numbers[left];
            tmp_pos = tmp_pos + 1;
            left = left + 1;
        }
        else
        {
            temp[tmp_pos] = numbers[mid];
            tmp_pos = tmp_pos + 1;
            mid = mid + 1;
        }
    }
    while (left <= left_end)
    {
        temp[tmp_pos] = numbers[left];
        left = left + 1;
        tmp_pos = tmp_pos + 1;
    }
    while (mid <= right)
    {
        temp[tmp_pos] = numbers[mid];
        mid = mid + 1;
        tmp_pos = tmp_pos + 1;
    }
    for (i=0; i <= num_elements; i++)
    {
        numbers[right] = temp[right];
        right = right - 1;
    }
}

```

التحليل التجريبي (Empirical Analysis)



شكل (14) كفاءة خوارزمية الدمج (Merge Sort Efficiency)

2- خوارزمية دمج الدمج المتوازنة (Balanced Two-way Merge Sort)

(Balanced Two-way Merge Sort)

- يمكن توصيف فكر هذه الطريقة بالمثل الأسى الخاص بكفاءة خوارزمية الدمج المتوازنة.
- 1 تقسم القائمة إلى قسمين متساويين تقريباً وتسمى A و B. ونضع كل عنصر من B مع نظيره الأول في القائمة A.
- 2 نقارن العنصر الأول في القائمة B مع العنصر نظيره الثاني في القائمة A ونضعه في القائمة C بالترتيب.
- 3 نقارن العنصر الثاني في القائمة B مع العنصر نظيره الثاني في القائمة A ونضعه في القائمة D بالترتيب.
- 4 نكرر الخطوات 2 و 3 لنضع على عناصر نظيرها 2 في كل من القامتين C, D ونضع العنصر بالترتيب في القامتين A, B.
- 5 نضع الطريقة نقوم بدمج عناصر القامتين A, B حيث عناصره بوضوح 4 لتكون مرتبة ونضعها في القامتين C, D.
- 6 نعيد الطريقة بدمج عناصر القامتين A, B بطول 8.
- 7 نستمر بهذا المنهج حتى الوصول إلى قائمة مرتبة.

مثال ٨: يقوم بترتيب العناصر ^١أ^٢ فيه باستخدام طريقة ترتيب المصنع ذو عنصرين
(Balanced Two way Merge Sort)

(18 , 23 , 2 , 50 , 42 , 63 , 20 , 28 , 33 , 47 , 3)

الحل : يمكن توضيح ذلك بالخطوات الآتية

1- N = 11 A = 18 , 23 , 2 , 50 , 42 ,
 B = 63 , 20 , 28 , 33 , 47 , 3

٢- ترتيب C = 18 , 63 2 18 , 42 47
 D = 20 , 23 , 33 50 3

تكون فائتين التاسمات لتوجيه D والتاسمات الفرعية C

3- A = 18 , 20 , 23 , 63 , 3 42 47
 B = 2 28 , 33 50 ,

4- C = 2 , 18 , 20 , 23 , 28 , 33 , 50 , 63
 D = 3 42 47

٥- A = ٢ 3 18 , 20 23 28 33 , 42 47 50 63 مرتبه

3- خوارزمية ترتيب المصنع باستخدام طريقة قسم وضم

(Divided and Conquer Merge Sort Algorithm)

يستخدم طريقة قسم وضم ولفصل في فرق قسم ، حيث تقسم المصفوفة المصفوفة إلى مجاميع فرعية
من التواليف. كل قائمة تكون من عنصرين بعد ذلك تقوم بدمج التواليف الفرعية المستخدمة ويمكن
بوصفها كما في المثال الآتي -

مثال // مستخدم طريقة فرق قسم في ترتيب المصنع المصفوفة التالية-

(43 , 54 , 11 , 41 , 93 , 17 , 5 , 66)

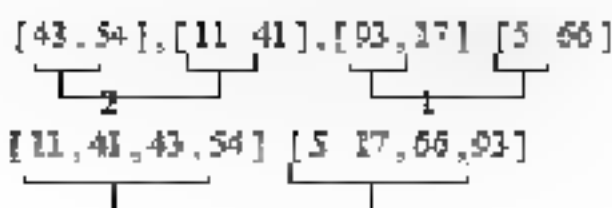
الحل : يمكن توضيح ذلك كما في الخطوات الآتية:

1- نجد عدد عناصر المصفوفة الرئيسية $N = 8$

2- نصفها إلى طوائف فرعية كل اثنين في قائمة

3- نقوم بدمج كل قائمتين بترتيب ودمجها

4- نعيد الخطوة السابقة للتوائم المتبقية بترتيبها



43 60 94 43 41 17 11 5 ختمه من قبله

هذه الطريقة هي ما يلي:

1- إذا كان لدينا مصفوفة A من الأعداد يمكن أن تكون أكبر من المصفوفة الأصلية ($n \times n$) عدد العناصر n .

2- إذا كان لدينا مصفوفة A من الأعداد $n \times n$ ، هذه المصفوفة A يمكن أن تكون $n \times n$ ، وهذا يعني أن الوقت المطلوب لإكمال عملية الترتيب كبير نسبة إلى n ، وهذا هو الفرق بين الطرق.

لا تستعمل من ذلك بأن

عدد المراحل $\log n$ No. Of Pass
عدد المقارنات $n \log n$ No. Of Comparison

الفصل الثالث

البحث

Searching

3.1 البحث (Search)

البحث هي عملية إيجاد عنصر معين في مجموعة من البيانات فإذا كان العنصر موجود في المجموعة العنصرية تفكير إيجابية وإذا لم يتكون سلبية في حالة عدم وجوده ، ولكن تكون العملية صالحة يحصل أن تكون العنصر مرتبة

3.2 البحث التسلسلي (Sequential Search)

في عملية البحث عن عنصر من خلال مسح أو استعراض عناصر القائمة من بدايتها وبالنسبة لحين الوصول لعنصر المطلوب إذا كان موجودا ما في حالة الوصول نهاية القائمة ولم يحصل عملية يعني أن العنصر غير موجود

عدد المقارنات: $n/2$

وقت التنفيذ: $O(n)$

مثال: البرنامج التالي يقوم بالبحث عن عنصر من بين مجموعة عناصر باستخدام البحث التسلسلي عفاً عن عدد العناصر (7) والعنصر المراد بالبحث عنه (3) والعنصر هو

data[] = 7,4,5,6,3,9,10

الحل: //

```
#include<stdio.h>
main( )
{
    int data[ ]={7,4,5,6,3,9,10};
    int nmax=7;
    int key=3;
    printf("%d\n",sqsearch(data,nmax,key));
}
Sqsearch(data,n,k),
int data[ ]
int n,
int k,
{
    Register int i
    For (i=0;i<=n,++i)
    If (k==data[ i ])return(i+1)
    Return(-1); /* no match exist */
}
```

نتيجة البحث هي أن العنصر (3) موجود في القائمة بالموقع (5)

3-3. البحث الثنائي (Binary search).

تقوم فكرة البحث الثنائي على تقسيم المصفوفة إلى نصفين واستبعاد النصف الذي لا يتلقى إليه المصباح key الذي يبحث عنه عن طريق تحديد العنصر الذي يقع في منتصف هذه المصفوفة، ثم يكرر هذا العنصر مع المفتاح الذي يبحث عنه (المصفوفة مرتبة ترتيباً) .
وال Pseudo code التالي يوضح لك هذه الطريقة.

```
repeat
If ID <= Array[k] then j=k-1
If ID >= Array[k] then i=k+1
until i>=j
If i >= j then we found the ID in the array
else the ID is not found
```

مثال : مصفوفة مرتبة ترتيباً أليحيى

```
word[] = {"begin", "const", "do", "end", "if", "odd", "program", "read",
"then", "var", "while", "write"}
```

كيف نكتب code محواريية البحث الثنائي ؟ كيف يبحث في المصفوفة بترتيب؟
نتم مقارنة عناصرها شيئاً عن طريق مقارنة قيمته المتساوية للقيمة في.

```
strcmp (char *str1, char *str2)
```

هذه الدالة تقوم بمقارنة حرفين أو مستطين حرفيين ونعود بإرجاع:

التيه (صفر) إذا كانت str1 = str2

قيمة سالبة إذا كانت str1 < str2

قيمة موجبة إذا كانت str1 > str2

و حرم البرنامج الذي نقوم بتطبيق خوارزمية البحث الثنائي (binary search) هو .

```
#include "STRING.H"
#include "STDIO.H"
#define max_size 12
//
int binary_search (ID)
char *ID
{
char *word[] = {"begin", "const", "do", "end", "if", "odd", "program",
"read", "then", "var", "while", "write"},
int i=0, j= max_size-1, s, k,
while(s<=j)
{
k=(i+j)/2;
s=strcmp(ID, word[k]);
if (s<=0) j=k-1,
if (s>=0) i=k+1,
}
```

```

if((i-1)>0){
    printf("have found the key (%s) at element %d", word[k] k+1);
    return k
}
return -1,
}
}
void main()
{
    int result;
    char *ID;
    printf("\nFiz. Enter the ID to begin search\nID=");
    scanf("%s",ID);
    result = binary_search(ID);
    if (result==1){
        printf("the key(%s) is not found" ID) }
    getch();
}

```

والتي تطبق خوارزمية البحث الثنائي (Binary Search) على مصفوفة م مبع الخسوف
المتخصصة التالية

- 1- الخطوة الأولى: وإذا لم بالنسبة لا يمكن تطبيق الخوارزمية إلا إذا كانت العناصر
مرتبة تصاعدياً أو تنازلياً أو فصحاً على حسب نوع العلاقات المتعددة فيها .
- 2- تحديد أول عنصر في المصفوفة الحرف 1 ، وآخر عنصر فيها والعرفاً مثلاً 7
- 3- تحديد العنصر الذي يقع في منتصف هذه المصفوفة الحرف 4
- 4- بعد ذلك يمكن تطبيق البحث الثنائي على مصفوفة إلى كين

أ- إذا كان يساويه يكون قد وجد العنصر الذي يبحث عنه
ب- إذا كانت قيمة المفتاح أقل من قيمة العنصر الأوسط في المصفوفة، فإن نحتاج أن نبحث
فقط في نصف المصفوفة الأول ونستبعد البحث في النصف الثاني
وإذا عاكس ذلك إذا كانت قيمة المفتاح أكبر من قيمة العنصر الأوسط في المصفوفة من نحتاج
أن نبحث فقط في نصف المصفوفة الثاني، ويستبعد البحث في النصف الأول

حيث نعتبر النصف الذي حسبنا البحث فيه مصفوفة قائمة بحددها، نحدد فيها $k, j, \& i$
(أي نقوم بتقسيمها إلى قسمين) ونطبق نفس الخطوات من 1 إلى 3 فيها، ثم نقرر الفتح مع
العنصر الأوسط الحدد نفس التوقيت الذي نكرر في الخطوة 1 إلى 3 السابقة.

أن خوارزمية هذا البحث تقوم بالبحث عن عنصر في قائمة مرتبة

- 1- تحديد موقع العنصر الذي يقع في منتصف القائمة تقريباً
- 2- إذا كان العنصر المطلوب (Item) مساوياً لعنصر في الوسط (X) فطبي ذلك أنه عملية البحث
مع العنصر الذي في الوسط انتهت، إذا كانت أقل من قيمة العنصر في الوسط نستبعد
البحث في النصف اليسرى و إذا كانت أكبر من قيمة العنصر (X) فطبي ذلك أن العنصر الذي يبحث عنه
أكبر من قيمة (X) فتكون البحث في النصف اليمى ولا نكرر

3 في الحالات عادية يتم المعالجة بنفس الطريقة التي نستخدمها في المعالجة بحسب الوصول إلى المحور المطلوب أي في حالة المحور هو N
عدد المتغيرات هو $\log_2 N$

```

}
}

void main()

int result;
int key;
printf("\nPlz. Enter the Key to begin search:\nKey=");
scanf("%d",&key);
result = binary_search(key);
if(result==1){
printf("the key(%d) is not found",key); }
getch();
}

```

عند م اء البحث في "ي مصفوفة عن عنصر محدد باستخدام بحث (Binary Search)

ان أقصى عدد من مر ثء البحث باستخدام Binary Search في ي مصفوفة يعطى من بحال القوة التي مرفح اليه رقم (2) مكي نصف العدد متى يريد عن عنصر المصفوفة هو حد، أي أنه لو قوة 1 (2) والي أعطي رقم أكبر من حد عناصر المصفوفة يربط

في مثال استخدم مصفوفة من (15) عنصر، ملاحظ أن بعد قمتي يريد على عدد عناصر المصفوفة هو حد، أي العدد (16) ناتج من القوة التي مرفح اليه رقم (2) هي (16=4*2)، مكني هذا يحتاج على أكثر أربع مرات مرفح في ال Binary Search حتى بعد العنصر الذي يبحث عنه، هن الممكن ان نجد هن، أو مة في العنصر أو نجد في ثاني مرة، أو ثالث مرة أو أربع مرة، أو لم يكن غير موجود في المصفوفة

عدد المقارنات هي $\log_2 n$ مثل، مكني القيمة العليا المطلوب البحث عن العنصر $key=29$ باستخدام طريقة البحث الثنائي (Binary Search)

9	11	16	18	25	29	32	35
---	----	----	----	----	----	----	----

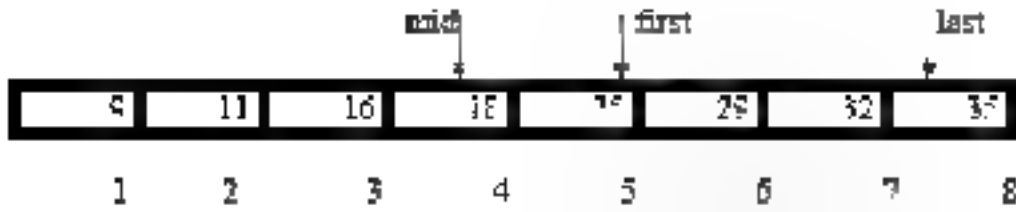
الحل: 1
 $I=1=first$
 $J=8=last$
 $Key=29$

1							
9	11	16	18	25	29	32	35
1	2	3	4	5	6	7	8

2- نجد القيمة الوسطية $mid = (1+8) \div 2 = 4$

$List[4] = 25$

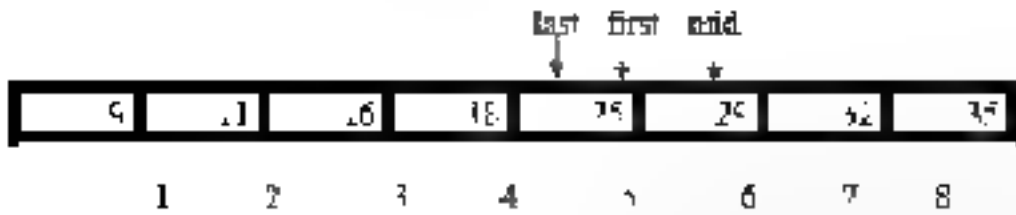
3 من يكرر $first = mid + 1$



4 نجد $mid = (5+8) \div 2 = 6$

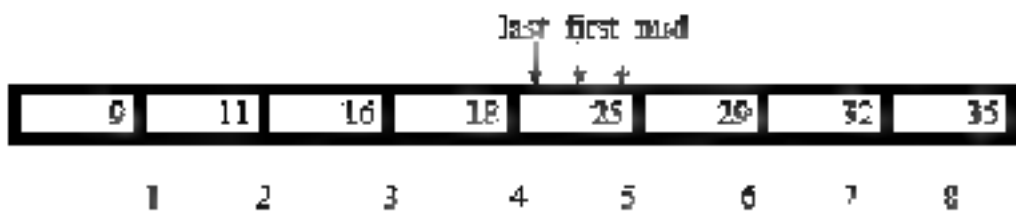
$List[6] = 26$

$Last = mid - 1$



5 نجد $mid = (5+5) \div 2 = 5$

$List[5] = 25$



النتيجة العنصر موجود بالموقع 5

مثال: استعمل الجدول التالي عن العناصر التالية بطريقة البحث الثنائي

Data[]=15,6,0,7 9,23,54 82,101

البحث / للعناصر المراد البحث عنها هي

X=101 x=14 x=82

X=101	Low=i=first	High=j=last	m.d
	1	9	5
	6	9	7
	8	9	8
	9	9	9

$m.d = (low + high) \div 2$

Found=101 في الموضع 9

X=14	Low=i=first	High=j=last	m.d
	1	9	5
	1	4	?
	1	1	1
	2	1	Not found

وسط التوقف هنا $low > high$

X=82	Low=i=first	High=j=last	m.d
	1	9	5
	6	9	7
	8	9	8

العناصر 82=موجود في الموضع 8

يفضل إيجاد معدل عدد المقارنات لكلية عناصر باستخدام القانون التالي.

Average of comparison=sum of comparisons÷number of elements

	1	2	3	4	5	6	7	8	9
element	15	6	0	7	9	23	54	82	101
comparisons	3	2	3	4	1	3	2	3	4

Average of comparison=25/9=2.77

4- البحث في الشجرة ثنائية (Binary Tree Search)

شجرة البحث الثنائية هي الشجرة التي كل عقدة فيها لكن من عقدها في اليسار عليها واحد من عقدها في اليمين عليها الشكل التالي صحيح ذلك.



شكل (S) : شجرة بحث ثنائية

تعد استخدام الشجرة الثنائية للبحث قائمة من الأعداد يمكن البحث عن عدد ما من خلال بحث العقدة التي بحرفه ونظيره وفق ما يلي.

إذا كانت العقدة المراد حذفها ورقية، يمكن حذفها دون تغيير بحر في الشجرة.

إذا لم يكن العقدة المراد حذفها ورقية، يجب عدم حذفها وتتمتع مكانها العقدة التي بحرفه العدد التالي وفقاً للترتيب المتصاعدي للأعداد المخرجة في الشجرة.

وجزاء البرنامج الخاص بتطبيق البحث في الشجرة السابقة هو

```

#include<iostream.h>
struct nodetype
{
    int k;
    struct nodetype* left;
    struct nodetype* right;
};
typedef struct nodetype* nodeptr;
nodeptr maketree(int x)
{
    nodeptr p;
    p=new nodetype;
    p->k=x;
    p->left=NULL;
    p->right=NULL;
    return (p);
}
  
```

```

void build(nodeptr node int number)
{
    if(number==node->k)
        if(node->right==NULL)
            node->right=makeTree(number);
        else
            build(node->right,number)
    else
    {
        if(number<node->k)
            if(node->left==NULL)
                node->left=makeTree(number);
            else
                build(node->left,number)
        else
            cout<<"Duplicate number "<<number<<endl;
    }
    return;
}

void search(int key nodeptr root)
{
    nodeptr p,f,q,rp,s
    p=root, // p will point to the node
    q=NULL, // and q to its father, if any.
    while(p!=NULL && p->k!=key)
    {
        q=p
        if(key<p->k)
            p=p->left
        else
            p=p->right;
    }
    if(p==NULL)
        cout<<"The key does not exist in the tree\n" //leave the tree
        unchanged
    else // rp will point to the node that will replace node p
        if(p->left==NULL) // node p has right son only
            rp=p->right;
        else
            if(p->right==NULL) // node p has left son only
                rp=p->left;
            else // node p has two sons

```



```

{
    f=p;
    rp=p->right;
    s=rp->left
    while(s!=NULL)
    {
        f=rp
        rp=s;
        s=rp->left; // s is always the left son of rp
    } // now rp is the inorder successor of p
    if(f!=p) // if p is not the father of rp
    {
        f->left=rp->right;
        rp->right=p->right;
    }
    rp->left=p->left;
}
if(q==NULL) // if p was the root of the tree
    root=rp;
else
    if(p==q->left)
        q->left=rp;
    else
        q->right=rp;
delete(p);
}

void print(nodeptr p)
{
    if(p!=NULL)
    {
        print(p->left);
        cout<<"p->key:"<<" ";
        print(p->right);
    }
    else
        cout<<" ";
}
}

```

```

void main()
{
    nodeptr tree;
    int number;
    cin >> number;
    tree = make_tree(number);
    while (cin >> number, number != 0)
        insert(tree, number);
    print(tree);
    cout << "Enter the number you want search for and delete:";
    int n;
    cin >> n;
    search(n, tree);
    print(tree);
}

```

اما جزء الخوارزمية الخاص بإزالة عقدة الشجرة البحث الثنائية فمكرر كالآتي:

```

T delete(T r)
{
    if (r == NIL)
        return r;
    if (r->key == x)
    {
        if (r->left == NIL)
            return r->right;
        else if (r->right == NIL)
            return r->left;
        else
        {
            if (key[x] < key[r->left])
                then z = delete(r->left);
            else z = delete(r->right);
        }
        r->left = z;
        if (y == NIL)
            then root = r;
        else if (key[z] < key[y]
            then left[y] = z;
            else right[y] = z;
        return r;
    }
}

```

y is maintained as the parent of x, since x eventually becomes NIL

The final test establishes whether the NIL was a left or right turn from y.

3- التحقق الرسمي في الحالات الأفضل والأسوأ والمتوسطة لخوارزمية (Beast & Worst & Average Complexity)

في المثال السابق دمج يتحقق في نفسه فقط، ولكن في معظم العمليات عند الانتهاء / مخرج
فقط بقيمة الحجم (n) ليس يتعلق أيضاً بالمتوى للبيانات المتعالجة لذلك مستخدم حقه للبحث
الاحتمالي عن عنصر في قائمة من العناصر حيث ان في هذه الحالة أربعة مقترحة عنصر x الذي
يبحث عنه مستخدم في القائمة المخزنة هي مصفوفة d وطولها المسجلة d = n، وإخراج مثال
العنصر في حال العثور عليه أو إرجاع القيمة -1 في حال عدم العثور عليه

مثال/ جزء من الجي ليصبح تطبيقاً بسيطاً عن عنصر x في قائمة مصفوفة d[]

```
int j = 0;
while (j < n && d[j] != x)
    j++;
if (j < n)
    return j;
return -1;
```

مثال / برنامج يحدد مفهوم التحقق Complexity على أن أفضل الأسلوب على التوجه في البرنامج
(قد لا تكون جميع هي x دقة 1)

```
{
    int mid;
    int first = 0;
    int last = N - 1;
    while (first <= last)
    {
        mid = (first + last) / 2;
        if (list[mid] == target)
            return mid;
        if (list[mid] > target)
            last = mid - 1;
        else first = mid + 1;
    }
}
```

في خطوات الحلقة الأفضل عنصر على عدد كبير لعنصر المطلوب إيجاد هو نفسه
عنصر المتصله، وبالتالي لنحتاج لتعقيد مقاربه وحيدة

أما التعيينات في البحث الأمثل إليها قد يحتاج لتعقيد رياضي لإيجادها فلو كان لدينا مجموعة من الأسماء موزونة ضمن معطى n كان في الشكل، والى البحث على اسم "جوزيف" فنتا نتيجة في ثلاث خطوات باستخدام محور رتبة البحث الثنائي، نحتاج متكرر بحجة إلى 7 خطوات لإيجاده باستخدام البحث الخطي.

begin				mid				end
0	1	2	3	4	5	6	7	
[محمد	علاء	مكي	موسى	أحمد	فوزي	عزیز	حسن
			begin			mid		end
					begin	mid		end
						begin	end	

ويمكن بعد العرف الى نتى في الخطوة `while` متالياً مع عدد العرف التي يجب على فيها تغيير المسألة n الحجم \times على 2 وبك حتى الوصول المسألة n في عنصر واحد .
تد وتنتهي بهذا العنصر، وهو العنصر المطلوب

في المثال كمرسمة من الأسفل 2 (نوعين متاليين) متالياً مت دوماً نعتمد المسألة الناتجة على 2 عنصر على متلته (مستوى) جديد
وبالتالي في التفرع في ذو التحويلات هو في التفرع في التفرع على 2 حتى نصل الى العنصر المطلوب في آخر خطوة العنصر هو لا يصح بدأ به وبالتالي مستطاح بين التفرع \log_2 خطوة بعد
والتي متلته سيكون 3 خطوات متتالية وجوز 8 عناصر

أما الخطوة المتخذ الوسطي فانه لا يمكن الوصول لمسألة نهائية فمتل، وجود العنصر في مكانه في التفرع هو لمتل وجود العنصر في المسألة الأصغر متالياً بمتل وجود في المسألة الفرعية التالية متروجا بمتل وجود في المسألة الفرعية التي بعدها وهكذا
بالتالي نجد في البحث الثنائي عدد يكثر من البحث الخطي وذلك لاقتصره عدد كبير جداً من العناصر في حال كانت n كبيرة بشكل كبير مثلاً عندما $n=100000$ فإن سوا بعد الحصول عليه في حالة البحث الخطي هو $n=100000$

أما في حال البحث الثنائي فإن السوا بعد سيكون $\log_2(100000)=16$ أي من البحث الثنائي وفر لنا 999,984 عملية مقارنة معاً للبحث الخطي

الفصل الرابع
الامتثالية في مسائل تصميم
الخوارزميات

**Optimization in)
Algorithms Design
(Equations**

1.4 المخطط (Graph)

المخطط عبارة عن مجموعة من العناصر التي تمثل نقاط وروابط (نمى (vertices) وروابط العناصر التي تمثل بعلاقات تسمى حواف (Edges) وهذه العلاقات تمثل بحدود كما في شكل رقم (1.6) التالي:



شكل رقم (1.6) يوضح مخطط غير متجه بسيط

$$G = (V, E)$$

$$V(G) = \{1, 2, 3, 4, 5, 6\}$$

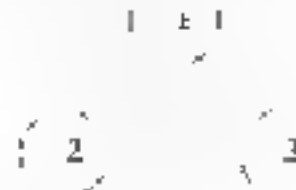
$$E(G) = \{(1, 2), (2, 3), (3, 4), (3, 5), (3, 6), (4, 5), (5, 6), (6, 3)\}$$

حيث V تمثل مجموعة من النقاط و E تمثل مجموعة من الحواف.

1.5 أنواع المخططات (Type Of Graphs)

يوجد نوعين من المخططات هي:

- 1 المخطط غير المتجه (Undirected Graph) هو المخطط الذي تكون العلاقة بين عناصره غير موجهة أي في الاتجاه تكون غير مهمة كما في الشكل رقم (1.7)



$$V(G) = \{1, 2, 3\}$$

$$E(G) = \{(1, 2), (2, 3), (3, 1)\}$$

شكل رقم (1.7) مخطط غير متجه

2- المخطط المتجه (Directed Graph).

هو المخطط الذي تكون الحافة بين عنصرين مرتبة، بمعنى أي من الإحداثيات تكون هوية، ومسبوقة كما في الشكل رقم (18).



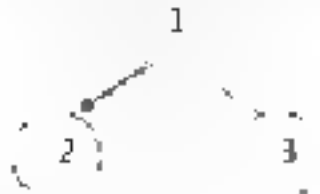
$$V(G) = \{1, 2, 3, 4\}$$

$$E(G) = \{(1, 2), (1, 3), (3, 4), (4, 3)\}$$

شكل رقم (18) مخطط متجه

2- المخطط المشترك (Undirected Graph & directed Graph)

هو المخطط الذي يحتوي على النوعين كما في الشكل رقم (19) التالي:



$$V(G) = \{1, 2, 3\}$$

$$E(G) = \{(2, 1), (1, 3), (3, 1)\}$$

شكل رقم (19) مخطط مشترك

العنصر هو مجموعة من المتغيرات التي تربط بين عطين في المخطط

مثلاً: في الشكل رقم (20) التالي يوجد عنصر بين نقطتين لهاته (1, 2)، (2, 3)، (3, 1).



شكل رقم (20) بوصف مخطط يجرى بهجوع عد مسارات

الخط :

1. المسار الأول للقطعة (1,5) هو: (1,3), (3,5) وليس (1,2), (2,3), (3,5)
2. المسار الأول للقطعة (1,6) هو: (1,3), (3,5), (5,6) وليس (1,2), (2,3), (3,6)

مع ملاحظة أنه لا يمكن كتابة المسار باستخدام هـ من المسار { }

3.4 طول المسار (Path Length) :

هو عدد المستقيمت (الخطوط) التي تربط نقطتين في المخطط كما نلاحظ في مخطط الشكل رقم 21

- أ. طول المسار بينه هو (2) ويمكن أن نجد بطول الآخر هو (3)
- ب. طول المسار بينهما هو (3) ويمكن أن نجد الطول الآخر هو (2)

والمعرفة طول المسار فإنه عند أن نصل النقاط من نصل عدد الزواج (عدد المستقيمت) في المخطط المسج أحيف يوجد أكثر من مسار بين نقطتين وبالتالي فإنه نعيد مشكلة تكرر في طول المسار (مسار متغير) كما في الشكل رقم (21) التالي.



1. (1,3), (3,4)
2. (1,3), (3,1), (1,3), (3,4)

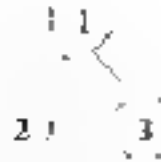
شكل رقم (21) بوصف مسارات مستقيمة الاتجاه

المخطط المتصل (Connected Graph)

هو المخطط الذي يوجد فيه مسار بين أية نقطتين من نقاط المخطط

المخطط غير المتصل (Dis-Connected Graph)

هو المخطط الذي تكون بعض نقاطه غير متصلة ببعضها البعض، يسمى بمسار أو الشظية رقم (22) التالي يوضح ذلك



مخطط متصل غير متجه



مخطط متصل متجه



مخطط متصل غير متجه



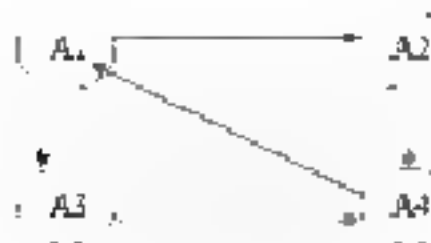
مخطط غير متصل متجه

شكل رقم (23) يوضح أنواع المخططات

يمكننا استخدام المخططات لتحديد أو معرفة أقصر المسارات من خلال إيجاد كل المسارات

وهي ثم نأخذ الإجمالي فيها

ممثل في الشكل المخطط التالي



المطلوب //

1. توضيح دور المخطط
2. تقسيم المخطط بهضمتين
3. بيان قيم الضغوط
4. بيان تأثير التغير في الخارجة من كل نقطة
5. إيجاد قصور مساري بين نقطة A1 ونقطة A4

الحل //

1. المخطط متجه وتمثل (Direct Graph & Connected Graph)
2. يمكن تغيير المخطط بالتصغير بالتالي

	1	2	3	4
1	A11	A12	A13	A14
2	A21	A22	A23	A24
3	A31	A32	A33	A34
4	A41	A42	A43	A44

3. يمكن بيان قيم المصفوفة أعلاه كالتالي

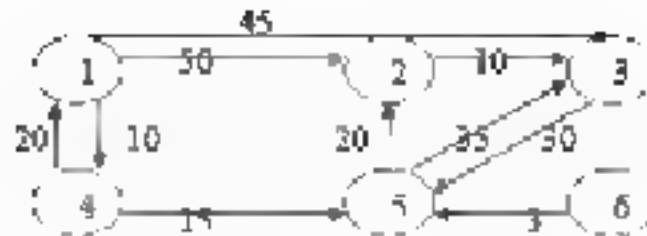
		A1	A2	A3	A4
2	A1	0	1	1	0
1	A2	0	0	0	1
1	A3	0	0	0	1
1	A4	1	0	0	0
		1	1	1	2

4. إيجاد مجموع تقييم الدخول والخارجة من كل نقطة كل
مجموع التقييم في كل صف يمثل عدد الخطوط الخارجة من كل نقطة
(Row = Out degree)
مجموع التقييم في كل عمود يمثل عدد الخطوط الداخلة للنقطة
(Column = Input degree)

السمات الداخلية	السمات الخارجية	اسم النقطة
1	2	A1
1	1	A2
1	1	A3
2	1	A4

5. إيجاد قصور مساري بين نقطة A1 ونقطة A4
a المسار (A2,A4) = (A1,A2,A4)
b المسار (A3,A4) = (A1,A3,A4)
وهو أقصر المسارات لأن درجتهما تساوي (2)

مثال 11 شبكة المخطط التالي



المطلوب:

- 1- حدد نوع المخطط
- 2- حدد المصفوفة
- 3- حدد قيم المصفوفة
- 4- حدد العيود الداخلة والخارجة
- 5- حدد المسارات بين نقطته
- 6- أعط القيم المسار هاتين النقطتين (1,5) و (1,6) و (2,6)

الحل:

- 1 المخطط متصل وموجه
- 2 المصفوفة هي

	1	2	3	4	5	6
1	A11	A12	A13	A14	A15	A16
2	A21	A22	A23	A24	A25	A26
3	A31	A32	A33	A34	A35	A36
4	A41	A42	A43	A44	A45	A46
5	A51	A52	A53	A54	A55	A56
6	A61	A62	A63	A64	A65	A66

- 3- حدد قيم المصفوفة كالآتي

	A1	A2	A3	A4	A5	A6
A1	0	50	45	10	0	0
A2	0	0	10	0	0	0
A3	0	0	0	0	20	0
A4	20	0	0	0	15	0
A5	0	20	35	0	0	0
A6	0	0	0	0	3	0

4. تحديد المسارات الداخلة والخارجة

المسار كـ الداخلة	المسار كـ الخارجة	سمم النقطة
20	105	A1
70	10	A2
50	30	A3
10	35	A4
49	55	A5
0	2	A6

5. إيجاد قصور مسار بين النقطتين

- المسار بين (1,6) هو " لا يوجد مسار بينهما
- المسار بين (1,5) هو (4,5) ، (1,4)
- المسار بين (2,6) هو " لا يمكن التوصل إلى النقطة (6) لعدم وجود مسار داخل إليها

4. طريقة الجشع (Greedy Method)

في هذه الطريقة نستخدم غالباً لحل مسائل الأمثلية (Optimization problems) التي علمه
 هي تكون (Maximum) ثلثي، معين أو تصغر (Minimum) غير الإمكان لتعين الشيء ، كيف
 في حلالة الربح أو الخسارة

في هذه المسائل نحاول على العنصر التالية-

- دالة هدف (Objective Function) وهي تمثل دالة هدف في نطاق معين. فكل حل ممكن
 ويكون حل ممكن، وسعى أفضل الحلول الممكنة وهو الأفضل .
- في مجموعة القيود (Constraints) جوان مجموعة الحلول التي يمكن اختيار
 ممكن حلول ممكنة (Feasible Solutions) في حل الممكن التي يعطي دالة هدف يساوي
 الحل الأمثل (Optimal Solution)

في الطريقة هذه الطريقة يمكن أن

يتم الحل الأمثل في طريقة الجشع على مراحل ففي كل مرحلة يُحدد أفضل قرار فبعض الخيارات
 أفضل منه ، بحيث في التمرير يمكن أن تكون الخيارات بحيث في تحقيق أفضل

ملاحظة: إن أغلب المسائل التي يمكنها طريقة الجشع تكون من أكثر من واحد إلى الفصل
 (n input) حيث ناهي يتلقى واحد من التعديل الذي حالياً هو الأمثل ولكن قد يكون هو لاحقاً
 قد فهذا يعني أنه يوجد مستوى أو مشاكل لهذه الطريقة .

يوجد نموذجين لطريقة الجشع هما

1. نموذج المجموعات الجزئية (Subset Paradigm)

في هذا النموذج يتم بناء مجموعة جزئية من العناصر من مجموعة عناصر معينة ويجب
 في تحقيق هذه المجموعة قيود المسألة ، وفيه يلي تجريد مبسط أو تحكمي لهذا النموذج .

```

SolType Greedy (type a[],int n)
a[1..n] contains the n inputs.
{solType solution=Empty, initialize the solution
For(int i=1; i<=n; i++)
{ typeX= Select(a);
  If feasible(solution,X)
    Solution= Union(solution,X);
}
return solution;

```

إن المقصود من (Solution) هي مجموعة العناصر التي لا تحتوي أي عنصر في المجموعة (Set S) فهي حالة يحتل بها مجموعة مختارة ونرجع واحد من العناصر لم يتم الاختيار هل إلى الحل هو حل ممكن؟ نعم فإنه يصلح هذا الحل لمجموعة الحلول السابقة وإلى حالة يمكن فإنه يضاف إلى الحل الأمثل

4.3- مسألة حقيبة (Knapsack Problem)

سنأخذ مسألة الجراب أو حقيبة الظهر كمثال نموذج للمجموعة التجريبية (Knapsack Problem) حيث هناك مجموعة كلف بوضع أي هذه الخفية

مجموعة العناصر (N) من الكيفيات حيث يمكن أن تكون أي شيء وأنت جراب مسطحة (C) مقاساً بالكيلو غرام (لاحظ كم مختلف وزن من هذه الكيفيات)

لتكن (I) مجموعة (N) حيث (N) حيث (1 ≤ i ≤ n)

بصفة الكمية (M) وهو يمثل حب المسألة حيث (0 ≤ K ≤ n) يحقق ذاته هي (P(I))

مختار - في الجراب بحيث نعلم الفائدة المحسنة أي أنه هناك مسأله (Maximum)

دالة الهدف هي معيار الأمثلية (نفسه الكمية بعد الدالة هي معيار الأمثلية أي الحل الأمثل)

$$\text{Maximize } \sum_{i=1}^n P_i X_i$$

ثم طباعة النتيجة (X) حيث بالاعتماد على قيمة النتيجة فإنه يمكن جلاء من الخلفه صنف أو لا صنف

إن توجد المسألة يجب أن نحقق

$$\sum_{i=1}^n W_i X_i \leq C$$

1. أي أي حل يحتوي ظهر المسألة هو حل ممكن

2. أي حل يحقق أكبر بعد زيادة الهدف هو الحل الأمثل

والنتيجة هي

$$0 \leq x_i \leq 1 \quad 1 \leq i \leq n$$

and

$$P_i \neq 0 \quad \forall \quad 0 \leq i \leq n$$

وسنقدم الآن مثالاً لتطبيق هذه الطريقة .

$$P[1, 3] = \{25, 24, 15\}, \quad C = 20, \quad n = 3$$

$$W[1, 3] = \{13, 15, 0\} \quad \begin{matrix} P_1 \\ P_2 \end{matrix} \quad (1, 3, 1 \neq 1, 5)$$

في هذه الطريقة تعطى ثلاث جدول عمقهم مختلف باختلاف العمق كما يأتي:
الخط الأول : أنه أول حل محتمره يكون هو الحل الأمثل بالقيمة
الخط الثاني : أنه أول حل محتمره يكون هو الحل الأمثل بالوزن
الخط الثالث : أنه أول حل محتمره يكون بالاعتماد على سعة تقسم الخفض على الواحد المقطرة .
سوف نقوم بعد جدول لتوضيح حلول المعادلة

المعامل	$\sum_{i=1}^n P_i$	$\sum_{i=1}^n W_i$	$x_1 \quad x_2 \quad x_3$
المعادلة الأولى بوزن	20	28, 2	$(1, 2/15, 0)$
الوزن الأقل أولاً	20	31	$(0, 2/3, 1)$
المعادلة الأمثل لكل وحدة وزن أي $(\frac{P_i}{W_i})$ أولاً	20	31, 5	$(0, 1, 1/2)$

وبالتوضيح هذه النتائج بتطبيق التعبير الأول كالتالي:

$$20 - 18 = 2$$

$$2 - 15 = 2/15$$

$$0 - 10 = 0, 10 = 0$$

في المعادلة (13) تعتبر قيمة واحدة وتساوي (1) بحيث بعد طرح الوزن من المعادلة يمكننا
نستخرج هذه المعادلة حتى يتم على الحروف
مع ملاحظة النتائج للخبر : أعلاه بالمحظوظ المعيار الثالث هو الذي يعطى بالنتيجة الغير قائمة
ومثالاً لنفهم (3, 2, 1) المعادلة له تعني الحل الأمثل.

ولذلك من صحة النتائج يمكننا إجراء عملية ضرب بين قيم المعيار النتيجة والوزن في
الخاصة بها .

وهذا هو الإجراء المتبع في تنفيذ خوارزمية Knapsack Problem

```

Void Greedy Knapsack (float m, int n)
// p[1..n] and w[1..n] contains profits and weights
// respectively of the n-objects ordered such
that  $p_i/w_i \geq p_{i+1}/w_{i+1}$ .
// m is the Knapsack size and x[1..n] is the solution vector
    n هو قسمة عدد الكائنات (m) على سعة الحقيبة ، ويجمع الكائنات بالترتيب
    من الأعلى إلى الأسفل ، حيث يصحح الترتيب تنازلي أثناء الحل وبالتالي فإنه إذا
    تمكنت من أخذ الأول والذي هو الأعلى سعة فإنه يبقى السعة في متغير x ، يكون خرج
    البرنامج أي يعطي كيف صغريه
{
    For(int i=1; i<=n; i++)    x[i]=0.0;
    float U=m;
    For(j=1; j<=n; j++)
    {
        If (W[j] > U) break
        X[j]=1.0;
        U = U - W[j];
    }
    If (j<=n)    x[j]=U/W[j];
}

```

مميزات الوقت للعبة

تطلب هذه الخوارزمية بعض الخطوات من وقت ترتيب الكائنات ابتدائياً $O(n)$ من الوقت فقط حيث إنه معطيات الوقت يكون صلب وقت الترتيب، حيث أنه في حالة استخدام خوارزمية ترتيب على وقت الترتيب هي $n \log n$ لأنها تكون دائماً وتعطي حلاً أمثل

ملاحظة: يوجد خوارزمية أخرى (Knapsack 0/1) حيث أنها هي بعض النتيجة أو لأحدها وبالتالي فإنه يجب حذف نظرية (If) الأخيرة من هذه الخوارزمية وفي هذه الحالة لا يكون هناك صغري الحسنة على حل أمثل

2. نموذج الترتيب (Ordering paradigm).

في هذا النموذج يتم اتخاذ القرارات باعتبار المسلمات بتوقيت معين بحيث كل قرار يتم اتخاذه باستخدام معيار التفضيل والذي يمكن حسابه من خلال تفرعات المسألة السابقة

ملاحظه ، في مثال التبعين هناك قبل واحد يحصل يودي إلى تغير الحل

لترتيب هذا النموذج سوف نستخدم معيار يسمى مسألة الحفاظ التبع الإجمالي **Optimal Merge Pattern** حيث يختص هذه المسألة بالعد هي التالية

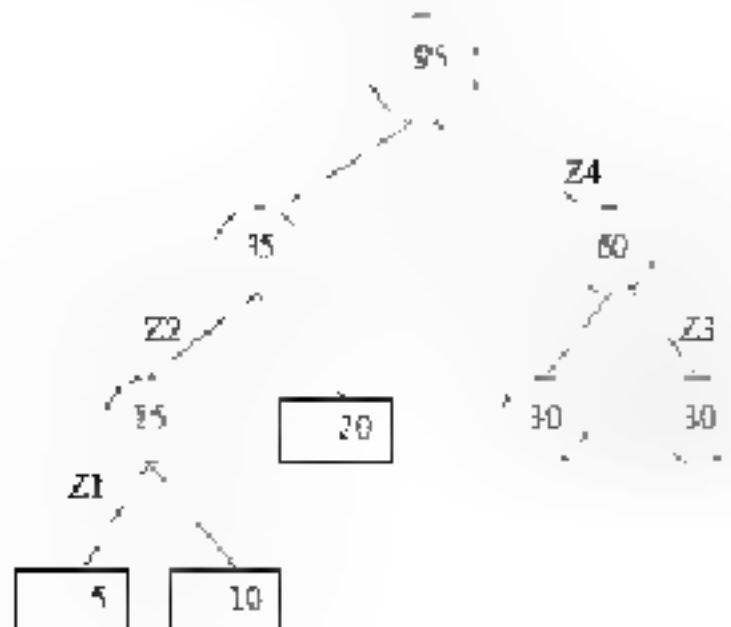
يوجد أربع مجموعة ملفات متخزون تمجيد يكون لدينا ملف واحد لكل وجناء عمليات ويمكن تحريك تقود الختمة بملفات المدمجة أي أنه ستكون مسالة (M-Number) .
 في هذه المسألة هي مسألة تفرج (D) من الملفات التفرجه على شكل ابروج ويزيد عدد واحد مرتب لكل عدد ممكن من الحركات السجلات ، في هذه المسألة تستدعي الترتيب هذا من تفرج من الملفات المراد تمجيد لتلك فهي تتعلق بنموذج الترتيب

في لاعداد المخرج هي

(التيكس حركة السجلات التفرج التفرج لأنك حجمًا مع كل خطوة أولاً).

مثال / يجب مجموعة الملفات المرصحة كما في التالي.

5 في (20 30 10.5,50) [F1 F5]



إن القيمة (20) تمثل عدد القنود في الشجرات في الصف الأول، وهكذا بالتسوية يتبعه 30 في القعدة (30) تمثل عدد القنود في الصف الثاني.
 في القيمة (10) تمثل طول الصف
 في ترتيب النسخ بهذه الترتيب هو الآخر.

$$20, 30, 15, 30 \\ 30, 30, 35 \\ 60, 35$$

إذا كانت W تمثل عدد الصفات الخارجيه للصف W في W يمثل طول الصف W
 في إلى بعد تكلي بحركات الشجرات لشجرة النسخ الثانية هذه تكون

$$\sum_{i=1}^n W_i \cdot W_i \\ = 5 \cdot 3 + 10 \cdot 3 + 20 \cdot 2 + 30 \cdot 2 + 30 \cdot 2 \\ = 205$$

وهذا هو الحد الأمثل الذي يكون هو الحجم الأمثل للصف
 نريد تطبيق عملية نسخ عشوائي.

وهذه هي الدالة التي تولد الشجرة للتطبيق مكتوبة بلغة C++.

```
Struct Tree node
{
    Struct tree* Lchild,*Rchild;
    Int Weight;
}
```

إن الجزء علامه حاصل بشكل العدة للشجرة الشافية بالقيمة الموصولة

```
Typedef Struct Tree node type;
```

```
Type *tree (int n)
```

List is global list of n single node binary trees as described above.

n تمثل عدد الصفات و W تمثل للقيمة W تمثل عدد القنود لكل صف

0	W	0	n
---	---	---	-------	---

```
For (int i=1, i<=n, i++)
```

```
{ type *pt=new type;
```

```
Pt->Lchild=Least (list);
```

```
Pt->Rchild=Least (list);
```

```
Pt->Weight=(Pt->Lchild->Weight)+(Pt->Rchild->Weight);
```

```
Insert (list,*pt);
```

```
}
```

إن هذا ال for يكون خاصه بتعبيره الدمج بين الصفات ، الدالة Least ترجع مؤشر إلى
 النقطة التي تكون ذات أقل وزن من الصفات ويحفظ مكانها ، الدالة insert تقوم بإضافة
 عنصر إلى القائمة list الخاصة بالعدد ، *pt هو رقم محطرات المؤشر ،

```
Return (Least(list));
```

```
}
```

إن الناتج من هذه العملية هو مؤشر إلى شجرة النسخ الثانية

و قمنا بالي توضيح موجد الخوارزمية

1. كل شجرة في القائمة List ستحتوي على راحة ، هذه الراحة هي عادة خارجة حيث تتكرر من ثلث جهات في (Rchild) ، (Wright) ، (Lchild).
(Rchild) و (Lchild) سيكونان هما مقربة ، أي الطفل (Wright) تحتوي على طرف بعد العلاقات التي تعطي
2. الدالة (Tree) ستعمل على تخزين شجرة (Least) و (Insert) حيث أن الدالة (Least) تقوم بإدخال شجرة في القائمة حتى إذا كانت لا تزال في القائمة فأنها ستبقى في القائمة وعوضاً عن ذلك يحذفها من القائمة
إذا الدالة (Insert) فإنها تقوم بإدخال شجرة في قائمة Pt إلى القائمة (Least) حيث أن مؤشري هذه نقطة هي Pt

3. في شجرة الترميز لتقليل التكلفة في هذه الخوارزمية نستخدم الخوارزمية لتقليل التكلفة من شجرة حيث يجرى الترميز على تلك الشجرة التي ستحتوي على أكبر في الشجرة

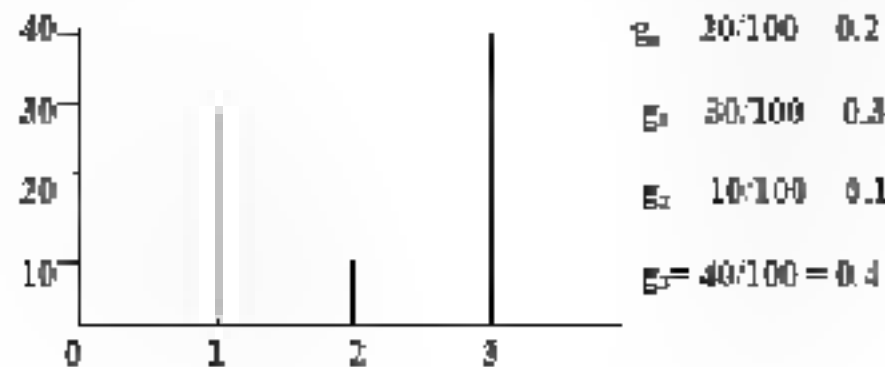
تعددت الوقت تخفيضه

إن هذه for الرئيسية تتكرر (n-1) من المرات ، في حالة الاحتفاظ بالقائمة (List) من حيث تعدد عدداً من حيث هي المرات في الحقول من القائمة (Least) تتكلف (O(1)) من الوقت والدالة (Insert) تعمل لتجربها بوقت هو (O(n)) أي أن الوقت الكلي المستغرق هو (O(n^2)).

4- استخدام قاعدة الترميز في شجرة ثنائية

طريقة هافمان (Huffman code) سبقت بمبدأ العالم هافمان 1952 ، تعتمد على فكرة تقليل البيانات وتصميمها بدون التغير على مدة الترميز أي بدون فقدان البيانات

هافمان / ليف هافمان طريقة هافمان تعتمد على Greedy mile مسألة بالمرجع التكراري الثاني



مثال ضم المبرج



ب. ترتيب القيم وجميع القيمتين



ج. الاستمرار بجمع بقية الأصغر 5 أصغر الرموز بقيمتين فقط

مقود الا عتياييه Original gray level (natural code)	الاحتمالية Probability	معدرة هرفمان Huffman code
$g_0 : 00_2$	0.2	010_2
$g_1 : 01_2$	0.3	00_2
$g_2 : 10_2$	0.2	011_2
$g_3 : 11_2$	0.4	1_2

يجد Entropy الخاص بالاحتمالية المستخدمة

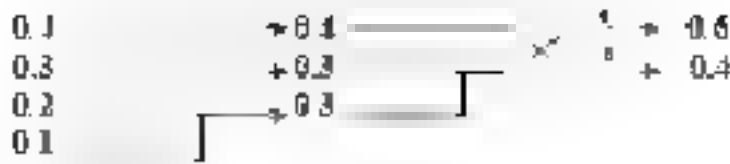
$$\begin{aligned} \text{Entropy} &= -\sum_{i=0}^3 P_i \log_2 (P_i) \\ &= [(0.2) \log_2 (0.2) + (0.3) \log_2 (0.3) + (0.1) \log_2 (0.1) + \\ &\quad (0.4) \log_2 (0.4)] = 1.846 \text{ bits/pixel} \end{aligned}$$

كما ان يوجد $\log_2 (X)$ يمكن الحصول عليه حسب القانون التالي:
 $\log_2 (X) = 1.322 * \log_{10} (X)$

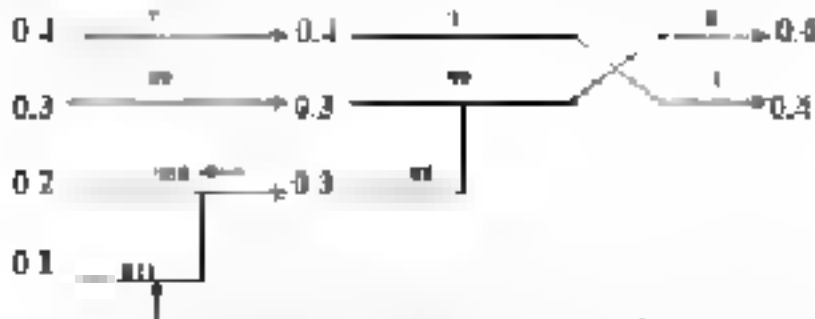
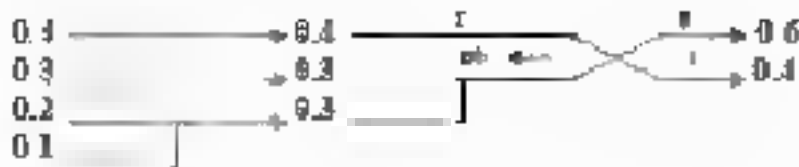
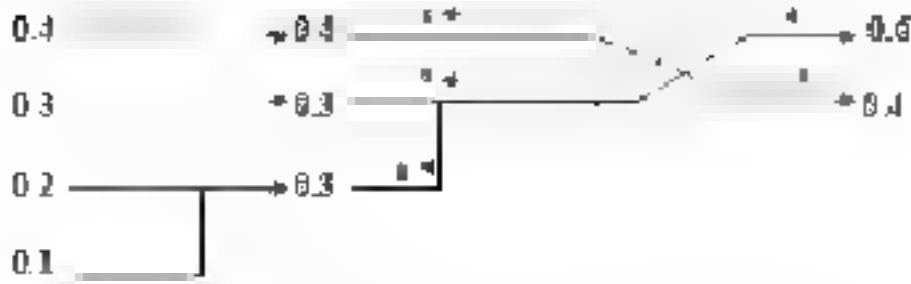
يجد معدل طول الشفرة حسب القانون التالي (Average length)

$$\begin{aligned} \text{Lave} &= -\sum_{i=0}^3 L_i P_i \\ &= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4) = 1.8 \text{ bits/pixel} \end{aligned}$$

مطور التفرع



مطور غير حجب



مثال: لنفرض اننا نستخدم أمثلة قاعدة المجموع لتحديد سرعة التفرع

Horizontal code Example (from our text)

Order	Probability	Order	Probability
1	0.4	1	0.4
2	0.3	2	0.3
3	0.2	3	0.2
4	0.1	4	0.1

Uniquely decodable			Not uniquely decodable			
Symbol	P_i	code	s	t	u	
a_1	$\frac{1}{4}$	1	0.1	0.1	0.1	Not uniquely decodable
a_2	$\frac{1}{4}$	01	0.3	0.3	0.3	
a_3	$\frac{1}{4}$	10	0.3	0.3	0.3	
a_4	$\frac{1}{4}$	001	0.1	0.1	0.1	
a_5	$\frac{1}{4}$	010	0.3	0.3	0.3	
a_6	$\frac{1}{4}$	100	0.3	0.3	0.3	

Although it might not look like it this is both uniquely decodable and instantaneous code
 Think about how you'd decode an incoming bit stream
Uniquely decodable

$$H = -(0.3 \log(0.3) + 0.3 \log(0.3) + 0.3 \log(0.3) + 0.1 \log(0.1) + 0.1 \log(0.1) + 0.1 \log(0.1))$$

$$H = 2.44$$

Average code length

$$E = 0.1(1) + 0.3(2) + 0.3(2) + 0.1(3) + 0.1(3) + 0.1(3)$$

$$E = 1.8$$

Since that $H < E$ as expected, there is no code that is better (longer) to find any code which did better (i.e. closer to the optimal entropy H). That is why the Huffman code is a 'compact' code

شجرة هوفمان مستخدمة لتشفير النصية -

الطريقة المستخدمة لتشفير كلاً من

- 1- نص، جعله نصية
- 2- حساب تكرار كل حرف في النص
- 3- شجرة «نص» تلك، يجمع تلك قيمتين لكل حرف
- 4- برقيش الشجرة بحيث كل حساب على اتجاه اليساري يعطى 0 وكل حساب على اتجاه اليميني يعطى 1.
- 5- كتابة شجرة هوفمان لكل حرف أو رمز بالنسبة هو خلال سبع حساب
- 6- إيجاد معدل الترميز و Entropy

مثال، لووجد شجرة هوفمان النصية التالية

dead bead safe decided dad. Dad faced a faced-ab Dad a
 decided, dad be back }

Space	a	b	c	d	e	f	.
1"	12	4	5	10	12	4	4

2 جمع اول هفتین یکل عرب

Space	a	b	c	d	e	f	.
7	12	4	5	19	12	4	4

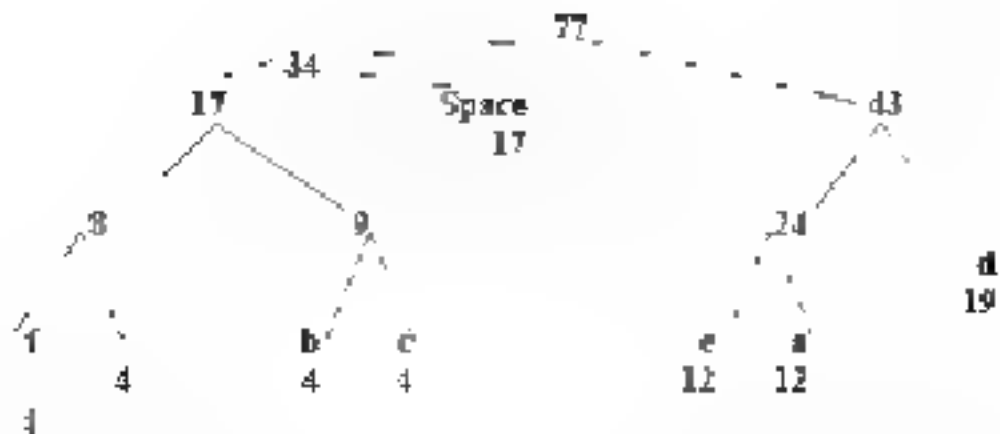
Space	a	b	c	d	e	f	.
17	12	4	5	19	12	4	4

Space	a	e	b	c	d	f	.
17	12	12	4	5	19	4	4

8	17	34	17	43	12
---	----	----	----	----	----

8	17	34	77	43	19
11	24	12	12	12	12

3. يرمز الشجرة



Space	a	b	c	d	e	f	.	رمز
01	101	0010	0011	11	101	0000	0101	شجرة هو

وحيث أن الخوارزمية الخاصة ببناء شجرة هو هافمان هي:

Huffman(C)

1. $n = |C|$
2. $Q = C$
3. for $i = 1$ to $(n - 1)$ do
4. allocate a new node z
5. $z \text{ left} = x = Q \text{ Extract-Min}$
6. $z \text{ right} = y = Q \text{ Extract-Min}$
7. $f[z] = f[x] + f[y]$
8. Insert(Q, z)
9. ...return $Q \text{ Extract-Min}$ return the root of the tree

حيث يمكن تقليل التعيد من $n \log(n)$ إلى n^2

الفصل الخامس
البرمجة الديناميكية
**Dynamic
programming**

1.5 البرمجة الديناميكية (Dynamic programming)

سمّ نفعلنا بمناج البرمجة الديناميكية بطريق مختلفة نكر في أني نموذج برمجي راسمي آخر . وعموما عن استخدام التتابع الفرعي و القيود يصف نموذج البرمجة الديناميكية الإجراء كيف هو وجهة نظر الحالة-والقرارات، والتحويلات، والفرجيف وهي

- 1- يبدأ إجراء من حالة ابتدائية حيث يتم اتخاذ قرار هـ
- 2- نسير في قرار . وانتقل إلى حالة حتمية
- 3- بالمرجع إلى الحالة الابتدائية، لحظة النهاية والقرار 1. يتم الوصول إلى قيمة مرجعة
- 4- نسير الإجراء غير مضمّن من الحالات إلى الوصول إلى حالة معينة

تكون مشكلة في إيجاد الحسنة التي تحصل القيمة تكلفة البرمجة 'محمّد' وتعدّ نموذج وسائط البرمجة الديناميكية أكثر ملاءمة للحلّ الذي يمر من تسلسل معين، يستند مقياس البرمجة الراسمة إليها لتتبع فاصد كثر في عمدا تكوّن معقولة بقرارات محدودة ومقطعة و عندما يكون التتابع الفرعي على خطي.

وقد وصف البرمجة الديناميكية بأنها الطريقة الإجمالية طرق الأمثلة مستقر على حلّ صلب واسع من المشاكل، يبين هذا النموذج مشاكل هجينة بشكل خاص حيث يمر بسير إلى آخر ذات حسابه فعلا، في تلك الحالات التي تتضمن تواضع غير مستقر، فيم مقطعة وهي هذه الحالات، قد تشكل البرمجة الديناميكية نهجية لنحلّ الوحيد للمشكلة

أو يعرف بين فاصد فاصد (Greedy method) التي تعتمد على خطوات علانية ويص البرمجة الديناميكية (dynamic programming) التي تعتمد على معلومات علمية مران في البرمي يتم بتعاقب قرارات واحد يتوقف النتيجة بوجود تعاقب قرارات كثيرة تكون القرارات التي تحوي قرارات جزئية غير مثلى لا يمكن أن تكون مثلى ولهذا لا تأخذ

فإن كان لدينا D من الخصائص لكل قرار من هذه $D \rightarrow n$ تعاقب قرار ممكن لنا فن حواريف البرمجة الديناميكية لها بتقيد مضمّن الحسنة

في 1 مثله على البرمجة الديناميكية

مثال // إيجاد الوقت الأمثل لـ n من القيم كالتالي

• a^i dominates a^j if $a^i > a^j$ since

$$\lim_{n \rightarrow \infty} a^i = 0 \quad \lim_{n \rightarrow \infty} a^j = 0$$

• $a^i + a^j$ dominates a^k since

$$\lim_{n \rightarrow \infty} (a^i + a^j) = 0 \quad \lim_{n \rightarrow \infty} a^k = 0$$

مثلاً، لحدة قفلة الوقت تقفلة هو نحو عدة لا من أقيم محددة كما هو موضح في الجدول الآتي:

n	$f(n) =$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = e^n$
10	0.01 μs	0.01 μs	10 μs	4.63 ms
20	0.04 μs	0.04 μs	101 μs	17 μs
30	0.09 μs	0.09 μs	1 sec	8.4×10^6 years
40	0.16 μs	0.16 μs	1.8×10^6	
50	0.25 μs	0.25 μs	13 days	
100	0.4 μs	10 μs	4×10^{30} years	
1×10^6	100 μs	1 ms		

3-5. تجميع البيانات (Data clustering):

تجميع البيانات هي عملية وضع البيانات في مجموعات مختلفة، خوارزمية التجميع تتقسم مجموعات من البيانات إلى عدة مجموعات، حيث أن التسايف إلى أنقطه من مجموعات معينة غير من المنطقيه بين بعض من مجموعات أخرى. فمثلاً، إلى فكر «تجميع البيانات هي فكر» بسيطة على فهمها وهي ثريه جدا من الأعداد في طريقة فكر «حيث أن كل مجموعة مع كليه كبيره من البيانات من إلى تخصيص الكم للكل من البيانات إلى عدد قليل من المجموعات والعاب، وبذلك من سحر تسهيل عمله التطبيق.

خوارزمية التجميع ستقوم على تقاطع واسع بين فقط للتطبيق وتحسين البيانات وإلا هي مقبولة لمعظم البيانات وبأكثر ج. ثريب البيانات، حيث أنه في كل مجموعة أو نجد مجموعات من البيانات وثقه بالإكمال بناء نموذج للمشكلة على أساس تلك المجموعات. هناك عدد من التقنيات المستخدمة في عملية تجميع البيانات، ومن هذه التقنيات (نحو ريف) التي سوف يتم الحديث عنها بشكل مفصل.

- K-means Clustering
- Subtractive Clustering

3. خوارزمية الـ (K-means Clustering)

هي خوارزمية تجمع عند من البيانات لتتخذ إلى خصائص ومميزات هذه البيانات، ومن عليه التجميع من حائل بغير «المسافات بين» أي «بالمركز» (centered cluster). وتقدم هذه الخوارزمية من حائل خطوات التالية:

1. حساب المسافات من مركز التجميع
2. حساب المسافات بين كل البيانات ومركز التجميع.
3. تجميع البيانات وتصنيفها في مجموعات بناء على أقل المسافات بين المركز ونقاط البيانات
4. اعتماد العديد الخطوات من 1 - 3 حتى الوصول إلى حالة التياب

يعتمد لدى هذه الحوزة على المراكز الأولية لمراكز التجمع (Centered)، ومن الممكن تنفيذ هذه الحوزة بعبارة هي اختلاف المراكز في كل مرة عن المراكز السابقة.

تفرض لدى أربعة أنواع من الأدوية، وكل نوع من الأدوية لديه عدد من المراكز في هذا المجال كل نوع من المراكز.

نوع الدواء (Medicine)	مؤشر الورق (Vaght Index)	أعداد المراكز (PH)
A	1	1
B	2	1
C	4	3
D	5	4

الهدف من هذا المجال هو جمع أنواع من الأدوية في مجموعتين اعتماداً على صفات كل نوع من الأدوية، وتحقيق هذا الهدف يمكن تنفيذ خطوات خوارزمية للتجميع كالتالي:

1. التقييم الإحصائي لمراكز التجمع.
تفرض أن الدواء A والدواء B هما من نفس التجمع الأولي، لكن A و C على اختلاف المراكز حيث أن $(A, B) \in C$ و $(A, C) \notin C$.
يبنى الشكل توزيع أنواع الأدوية المعبر عنه بالمعيار α على المستوى البيكاردي، كما يبين مراكز التجمع الإحصائية مع المراكز بعينها، أي هذه المراكز قد يبنى عليها بشكل عشوائي.

2. المسافات بين النقاط والمراكز.
نحسب المسافة بين مركز التجمع وكل نقطة من النقاط في المستوى لننتج عدد مصفوفة من المسافات، حيث في كل عمود في مصفوفة المسافات نكتب نوع الدواء و حسب الصف الأول من مصفوفة المسافات يتكون من المسافات بين كل نقطة ومركز التجمع الأول، والصف الثاني يتكون من المسافات بين كل نقطة ومركز التجمع الثاني.

3. جميع النقاط.
حيث نحصل كل نقطة بين مركز التجمع بالاعتماد على كل مسافة، وهكذا فإن الدواء (A) يذهب إلى المجموعة الأولى، الدواء (B) إلى المجموعة الثانية، الدواء (C) إلى المجموعة الثالثة، والدواء (D) يذهب للمجموعة الثانية.
يخرج لدينا مصفوفة المجموعات G التي تتكون من القيم 1 و 0 ويكون العنصر في مصفوفة المجموعات يساوي 1 فقط إذا كان الدواء مسدداً إلى تلك المجموعة.

4. التكرار الأولي: تحديد مراكز التجمع.
بعد مرحلة العناصر كذا المجهول، نحسب مركز جديد لكل مجموعة اعتماداً على هذه المصفوفات الجديدة المصنوعة الأولى لتكون من عناصر واحد فقط ونفسي إحداثيات مركز التجمع الأول، كما هي دور تحديد (A, B) و (C).
أد بالمجموعة الثانية والتي ستكون من قاعدت العناصر بتغير إحداثيات مركز التجمع الثاني بناءً اعتماداً على إحداثيات العناصر الخاصة.

تم حساب الفرق الأول $sc1$ والذي كان كثافة الحواف عند أعلى حافة من D_{sc} بعد ذلك يتم حساب قيم الكثافة بجمعية بعد كل نقطة sc .

وتقوم خوارزمية الـ *Subtractive clustering* بالخطوات التالية

1. إيجاد نقطة معينة موجودة في المجال تكون عدد الحواف عالية ويتم حساب الكثافة من المعادلة الأولى وهر تم اختيار نقطة معينة كمرکز ، وبذلك عن طريق وجودها بين عند كثير من النقاط المحيطة .

2. يتم حذف نقاط النسيج

3. يتم تبحث الخوارزمية عن مركز جديد وذلك عن طريق حساب قيم الكثافة للنقاط الأخرى كما في المعادلة الثانية ، وتستقر هذه العملية حتى لا يتغير من كل النقاط أو يوجد عدد كثيف (مجموع) من النقاط عند

بعد إجراء هيرش هذه الخوارزمية هي لها بكثر فعالية من الخوارزمية التي تكونت سابقا كما أنها الأسرع في أشكال المعطيات

4-5 خوارزمية (Dijkstra)

يخترع ديكسترا *Edsger Dijkstra* هو أحد العلماء الهولنديين في علوم الحاسب ولد في 1919م في مدينة روتردام ، ولد مشغولاً بالهندسة المعمارية في جامعة بيرن ، كان من علم ما أتت له أن الهندسة مصطب في علوم الحاسب

استلم ديكسترا عام 1972م جائزة *J.M. Turing* على تطوير مساهمته الأساسية في برمجة اللغة كما حظي بمناصبه في (كرسي سكرتيرجورنسون في نظام الحاسب) في جامعة تكساس في بوسطن منذ عام 1984م وحتى تقاعد عام 2000م

من أبرز إسهاماته في علوم الحاسب هي خوارزمية الخوارزمية لأقصر المسارات وأيضاً بحوارزمية ديكسترا ، مصطب هذه الخوارزمية في تنظيم نقل المعلومات بين أجهزة الحاسب وعرف كيف بعد خوارزمية الطرق لأقصر المسارات

كتب ديكسترا عام 1958م ورقة بحثية هامة في تخصصات أنظمة البرمجة الحاسب وعملات النماذج السيمولية وانتشر بحثاً بديكسترا ، عبارة البرمجة شوبيرد (تسمى في أكثر تستخدم تلك بـ "I more use a for 2") والتي تعبر إلى حقيقة أنه يجب نجد نفسك تقدم أكثر من عمل تجيبه معاً عليه لأنه حتى يوجد سلك من هذا المسار داخل حدة مركزية

والخوارزمية للحصول على الطريقة هي

$DIJKSTRA(G, w, s)$

1. INITIALIZE_SINGLE-SOURCE(G, s)
2. $S \leftarrow \{ \}$ S will ultimately contain vertices of final shortest-path weights from s
3. Initialize priority queue Q i.e. $Q \leftarrow V[G]$
4. while priority queue Q is not empty do
5. $u \leftarrow EXTRACT_MIN(Q)$ // Pull out new vertex

```

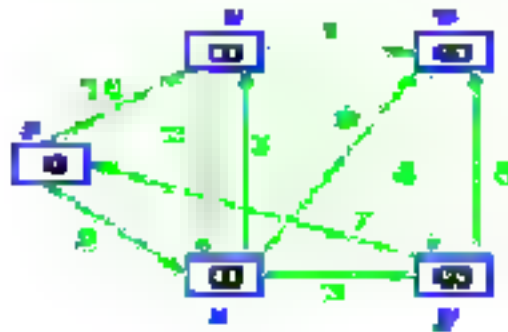
6   S ← S ∪ {u}
      Perform relaxation for each vertex v
      adjacent to u
7   for each vertex v in Adj[u], do
8     Relax (u, v, w)

```

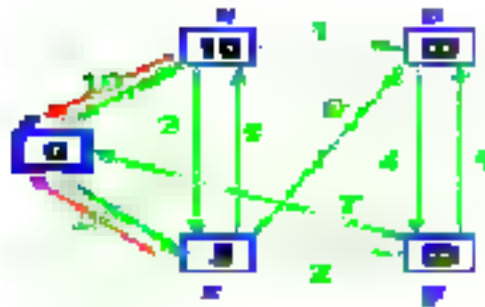
5.5 بناء تطبيق خوارزمية (Dijkstra)

مثال // تطبيق الخوارزمية كما في الخطوات التالية

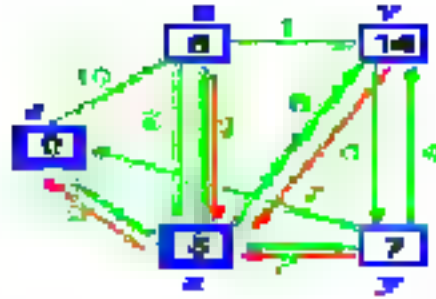
1 الاسم مستطد $G=(V, E)$ كل قبة هناك كات عبر سوية عا العدة S حيث كات Q



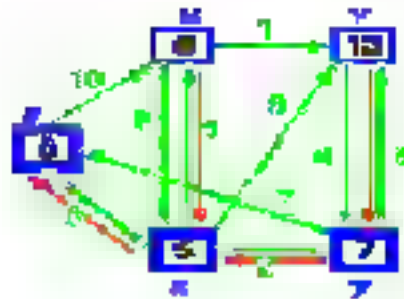
2 أولا نحدد عقدة أوله من S ونحدد $d[S]$



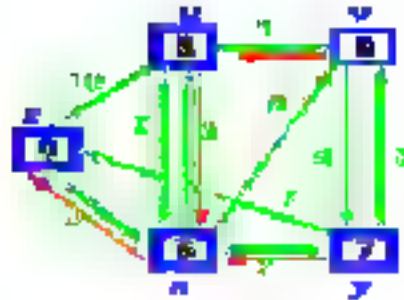
3 الأخير عدة x ونطبق خطوات الخوارزمية



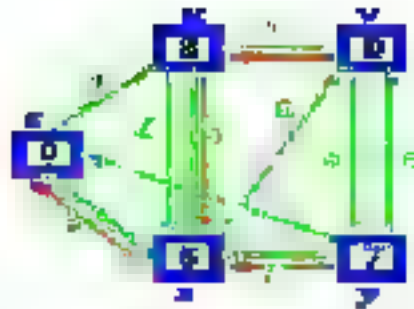
4. اكتب ψ قيمة البعد ψ وبتطبيق هذه الخطوات



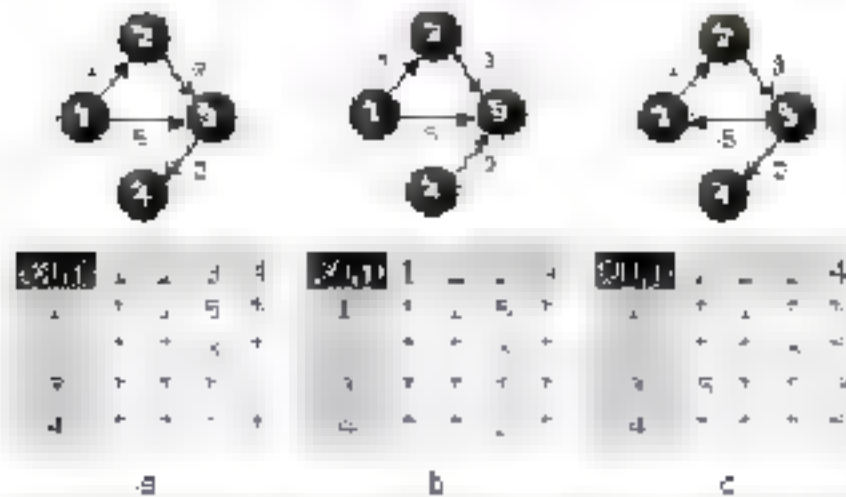
5. الآن هناك المقام ψ نحرك عقدة ψ جانبها V



6. عقدة ψ المسقط سوف يعطى أصغر مسار ψ نغير تصنيف العقدة



مثلاً : نقوم بتطبيق الطريقة المقترحة ليترك لنا المخطط الآتي



let $C = \{1, 2, \dots, n\}$ denote the set of cities and for each city j in C let $P(j)$ denote the set of its immediate predecessors and let $S(j)$ denote the set of its immediate successors, namely set

$$P(j) = \{k \in C \mid D(k,j) < \text{Infinity}\}, j \in C$$

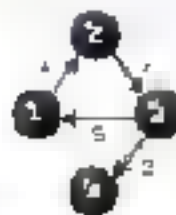
$$S(j) = \{k \in C \mid D(j,k) < \text{Infinity}\}, j \in C$$

Thus, for the problem depicted in Figure 1 a) $P(3) = \{0\}$, $P(2) = \{0\}$, $P(4) = \{0, 1\}$, $P(5) = \{0, 1, 2\}$, $S(0) = \{1, 2, 3\}$, $S(1) = \{4, 5\}$, $S(2) = \{5\}$, $S(3) = \{6\}$, $S(4) = \{6\}$, $S(5) = \{6, 7\}$ where $\{\}$ denotes the empty set

Exercise 10. If \mathcal{D} depicts the net of \mathcal{P} , then, for every node i in \mathcal{D} , denote by \mathcal{D}_i and \mathcal{D}^i the subnets of \mathcal{D} that have i as terminal place and initial place, respectively.

$$\mathcal{D}_1 = \{ \}, \text{ and } \mathcal{D}^1 = \{1\}$$

$$\mathcal{D}_5 = \{ \}, \text{ and } \mathcal{D}^5 = \{1\}$$



$\mathcal{D}(\mathcal{D}_1)$	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

a



$\mathcal{D}(\mathcal{D}_5)$	1	2	3	4	5
1	*	0	*	*	*
2	*	*	1	*	4
3	*	*	*	3	4
4	*	5	*	*	2
5	*	*	*	*	*

b

Matrix	1	2	3	4	5	6	7	8	9	10
1	*	1	*	*	*	*	*	*	*	*
2	*	*	3	*	*	*	*	*	*	*
3	-5	*	*	2	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*
10	*	*	*	*	*	*	*	*	*	*

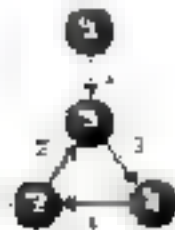
يتم هذا إيجاد الصور المتساوية

x_{ij} Quantity (Flow) sent along the link from city i to city j
 $i=1, \dots, n$

then the minimum cost network flow problem is as follows

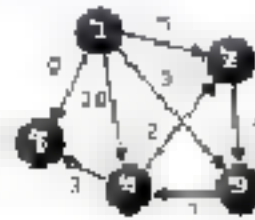
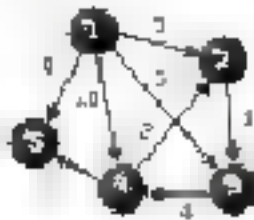
$$\begin{aligned} \min \sum_{i,j} f_{ij} x_{ij} \\ \text{s.t.} \quad \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} = a_i \quad i=1, \dots, n \end{aligned}$$

مثال: توضيح الطريقة



$$\begin{aligned} f_{12} &= 0 \\ f_{13} &= 4 + f_{43} \\ f_{24} &= \min \{2 + f_{42}, 1 + f_{14}\} \\ f_{34} &= 3 + f_{43} \end{aligned}$$

مشاركون



f_{ij}	2	3	4	5
1	4	3	10	9
2	5	2	1	1
3	4	2	4	1
4	1	2	4	1
5	1	1	1	1

a

f_{ij}	2	3	4	5
1	4	3	10	9
2	5	2	1	1
3	4	2	4	1
4	1	2	4	1
5	1	1	1	1

b

الص

Iteration k	J	F	Iteration k	U	F
0	$\{1, 2, 3, 4, 5\}$	$(0, x, x, x, x)$	0	$\{1, 2, 3, 4, 5\}$	(x, x, x, x, x)
1	$\{2, 3, 4, 5\}$	$(1, 5, x, x, x)$	1	$\{2, 3, 4, 5\}$	$(1, 5, x, x, x)$
2	$\{3, 4, 5\}$	$(2, 5, 2, x, x)$	2	$\{3, 4, 5\}$	$(1, 5, 2, x, x)$
3	$\{4, 5\}$	$(0, 5, 3, 7, 9)$	3	$\{4, 5\}$	$(0, 5, 3, 7, 9)$
4	$\{5\}$	$(0, 5, 3, 2, 8)$			
	a			b	

$$F_{k+1} = F(5) = 9 \times F(a) = F(b) = 7.$$

الخوارزمية لتابعه العتال تكون كالآتي

```

Initialize  $k=1$   $F(1) = 0$   $F(\infty) = \text{Infinity}$   $j = \infty$ 
 $U = \{1, 2, 3, \dots, n\}$ 
Iterate While  $(U \neq \emptyset \text{ and } F(j) < \text{Infinity})$  Do
     $u = \min(U)$ 
     $F = \min(F(j), D(u) + F(u+1))$   $j = u$ 
     $U = \arg \min \{F \text{ in } U\}$ 
End Do

```

Range	Sparsity	Acyclic	$D(i,j) \leq 0$	Gen					
File	Status	Generated new problem	New algo	Solve Help					
$N \times N$?	?	?	?	?	?	?	?	?
$DIP(i,j)$?	?	?	?	?	?	?	?	?
$D(i,j)$?	?	?	?	?	?	?	?	?
$D(i,j)$?	?	?	?	?	?	?	?	?
			4	5	6	7	8	9	10
1	6	6	10	7	8	5	3	2	
2	3	5	7	7	0	7	9		
3		7	6	3	0	4	6		
4			4	7	0	4	7		
5				10	5		3	5	
6						4	6	8	
7						6	9		
8							9		
9								8	
0									

هناك استخدام خوارزمية ديناميكية في حركة الأربوب ويجب أن تكون متداولة

Example arena:

```

X-----
B   B
B-----
B-----
  BB-
---B---
      B.B
---B-B
B-----Y

```

Input

```

8 8
.0
. 1
. 6
2 0
3 2
4 4
4 5
5 4
6 5
6 7
7 0

```

Sample Output:

```

0 0
0 1
0 2
. 2
2 2
2 3
2 4
2 5
2 6
3 6
4 6
5 6
6 6
7 6
7 7
. 1

```

PROGRAM (TRIED AND TESTED IN TURBO C++

```
#include <stdio.h>
#include <string.h>
struct Matrix { short int **array,
                int row,int col;
                };
struct Vertex { int num; int curDist
                };
typedef struct Matrix matrix;
typedef struct Vertex vertex;
void getGrid(matrix &m)
void getShortestPath() /* Dijkstra Algorithm */
void printSolution,int p[],int index);
matrix m;
int main()
{
    getGrid(m)
    getShortestPath()
    printf("n 1 1'm a");
    free(m.array);
    return 0;
}
void getGrid(matrix &m)
{int ctr1,ctr2 blockedSquares,x,y
scanf("%d%d%d%d",&m.row,&m.col,&blockedSquares)
m.array=(short int **)malloc(m.row*sizeof(short int *)),
for(ctr1=0;ctr1<m.row;ctr1++)
m.array[ctr1]=(short int *)malloc(m.col*sizeof(short int));
for(ctr1=0;ctr1<m.row;ctr1++)
for(ctr2=0;ctr2<m.col;ctr2++)
m.array[ctr1][ctr2]=0;
for(ctr2=0;ctr2<blockedSquares;ctr2++)
{
    scanf("%d%d",&x,&y);
    m.array[x][y]= 1
}
}
```

```

void getShortestPath() /* Uses Dijkstra Algorithm */
{
    int ctrl = 0, ctr2 = 0, row1, col1, row2, col2,
    int *predecessor = (int *)malloc(sizeof(int) * mrow * mcol);
    vertex *toBeChecked, minVertex,
    toBeChecked = (vertex *)malloc(sizeof(vertex) * (mrow * mcol + 1));
    for (ctr1 = 1; ctrl <= mrow * mcol; ctrl++)
        predecessor[ctrl] = 3.000;
        toBeChecked[ctrl].num = ctrl - 1,
        toBeChecked[ctrl].currDist = 3.000;

    predecessor[0] = 0;
    toBeChecked[0].num = toBeChecked[0].currDist = mrow * mcol;
    toBeChecked[1].currDist = 0;
    while (toBeChecked[0].num != 0)
    {
        minVertex = toBeChecked[1], ctr2 = 1;
        for (ctr1 = 1; ctr1 <= toBeChecked[0].num; ctr1++)

            if (toBeChecked[ctr1].currDist < minVertex.currDist)
                {ctr2 = ctr1; minVertex = toBeChecked[ctr1]}
            },
        toBeChecked[ctr2] = toBeChecked[toBeChecked[0].num];
        toBeChecked[0].num =
        row1 = minVertex.num / mcol; col1 = minVertex.num % mcol;
        for (ctr1 = 1; ctrl <= toBeChecked[0].num; ctr1++)

            row2 = toBeChecked[ctr1].num / mcol;
            col2 = toBeChecked[ctr1].num % mcol;
            if (marray[row2][col2] == 0)
            if (((col1 < col2) * (col1 < col2) == 1 && row1 == row2) || ((row1
                row2) * (row1 - row2) == 1 && col1 == col2))
            if (toBeChecked[ctr1].currDist > minVertex.currDist + 1)

                toBeChecked[ctr1].currDist = minVertex.currDist + 1,
                if (toBeChecked[ctr1].num == 0)
                predecessor[toBeChecked[ctr1].num] = minVertex.num;
            }
    }
}

```



```

printSolution(predecessor m,row*m.col-1);
}
void printSolution(int p[],int index)
{
    if(index==0)
        printf("0 0");
        return;
    };
    f(p[index]==31000)
        return;
    printSolution(p,p[index]);
    printf("%d %d",index/m.col,index%m.col);
}

```

6-5 المسحوظ متعدد المراحل (Multistage graph)

هو مسطح موجبة قيمة تقسم بعدد إلى $(K \geq 2)$ المجموعات منفصلة (V_1) حيث $(1 \leq i \leq K)$ المجموعة (V_i) تتكون من حيث $(|V_i| \geq 1)$ (أي أنه عند العناصر الموجودة $(i=1)$ حيث i المجموعة الأولى والثانية تحتوي على عقد واحد

أقر من S إلى t في عقدة البداية في V_1 وإلى t في عقدة النهاية في V_K وأقر من V_i إلى V_{i+1} في كل عقدة (i, r) هناك كلفة الحافة بين (i, r) كما في المعادلة التالية

$$C(i, r) = \min_{j \in V_{i+1}} \{ c(i, r, j) + C(i+1, j) \}$$

كف يرمز للحافة الموجودة بكل مسطح (i, r) إلى كلفة المسار من البداية S إلى النهاية T في مجموع كلف الحواف على المسار

ملاحظة 1: مسلة المسطح متعدد المراحل في إيجاد مسار أقل كلفة من $(S$ إلى $T)$ كل مجموعة V_i تمثل مرحلة في المسطح وكل مسار من $(S$ إلى $T)$ يندرج في مرحلة 1 وينتهي بالمرحلة K .

$$Cost(i, r) = \min_{r \in J_i + 1 \leq j \leq E} \max \{c(i, j, r) + \cos t(j+1, r)\} \quad 1$$

وهي كلفة المسار الأقل كلفة حيث

في ذلك $\langle r, r \rangle$ هي حكة تنتمي إلى جميع الحواف الموجودة في المخطط بحيث

$$Cost(k+1, j) = \begin{cases} c(j, j) & \text{if } j \leq j \leq E \\ \infty & \text{if } j < j \leq E \end{cases}$$

ولهذا فإن المعقنة رقم (1) تقطع للحالة $Cost(2,5)$ بصفاً إلى

$$Cost(k+2, r), \forall r \in F_{k+2}$$

تتبع ذلك

$$Cost(k+3, j), \forall j \in F_{k+3}$$

وهكذا نسير على أن نصل في فقط الموجود في $P(0)$ وبغاية $Cost(2,5)$ حيث نجد
أخرى قيمة للمسار الأقل كلفة

ويعبر المخطط الموجود في الشكل رقم (2) السابق بصل على

$$Cost(3,6) = \min\{5 + \cos t(4,9), 5 + \cos t(4,10)\} = 7$$

وهي قيمة المسار الأقل كلفة

$$Cost(3,7) = \min\{4 + \cos t(4,9), 3 + \cos t(4,10)\} = 7$$

$$Cost(3,8) = \min\{5 + \cos t(4,10), 6 + \cos t(4,11)\} = 7$$

$$Cost(2,2) = \min\{4 + \cos t(3,6), 2 + \cos t(3,7), 1 + \cos t(3,8)\} = 1$$

$$Cost(2,3) = \min\{2 + \cos t(3,6), 7 + \cos t(3,7)\} = 9$$

$$Cost(2,4) = \min\{11 + \cos t(3,8)\} = 18$$

$$Cost(2,5) = \min\{11 + \cos t(3,7), 8 + \cos t(3,8)\} = 15$$

$$Cost(1,1) = \min\{9 + \cos t(2,2), 7 + \cos t(2,3), 3 + \cos t(2,4), 2 + \cos t(2,5)\} = 6$$

لاحظ أن الحد للكلف الأقل بصورة متتالية دائماً لأنه الأصغر حيث أن كل كلفة مسارات
صغيرة هي كلفة $Cost(1,1)$ وتعني كلفة $Cost(1,5)$

ويبدأ من المسار الأقل كلفة هو S إلى T ، كان بأكمله متعارفاً 16 واختياد المسار بسجل
المراتب المستخدمة في كل حالة (عند)

ونفترض أن $D[i]$ هي قيمة T التي تصغر العلاقة التي ذكرناها سابقاً وهي

$$c[i, r] + \text{Cost}(\{i + 1, r\})$$

ملاحظة: // في $D[i, r]$ يمثل القرار المسند للمرحلة القادمة حيث ذكرى بحفظه التي نحفظه في
قيمة مصب العلاقة السابقة

ملاحظة: // في قدم القرار r المسند هي نفس الرقيم للخط الموجوده على المسار ورائف بدلاً
بالمسار (2-2)

$$D[2,2] = 7, D[2,3] = 6$$

$$D[2,4] = 8$$

$$D[2,5] = 8$$

$$D[1,1] = 7$$

وباعتبار المسار الأقل كلفة هو

$$S = 1, V_2, V_1, \dots, V_3, T = 12$$

حيث

S تمثل النقطة الأولى وهي الأقل كلفة

$$7 = V(2)$$

$$7 = V(3)$$

$$10 = V(4)$$

$12 = T$ وتمثل النقطة الأخيرة

$$V_2 = D[1,1] = 7$$

$$V_1 = D[2, D[1,1]] = D[2,7] = 7$$

$$V_3 = D[2, D[1,1]] = D[3,7] = 10$$

ملاحظة: // يجب تحديد النقطة الأولى بغير إلى ذلك الموال التالي ما هي النقطة الأقل التي يجب

أن نختارها في المرحلة التالية

مستخرج من المسار الأفضل هو $(T) 12 10 7 2 S$

والآن سوف نقوم بكتابة الخوارزمية لحل هذه المسألة (خوارزمية المخطط متعدد المراحل

المسطرة لتطبيقه الصماعة)

تتضمن خوارزمية المخطط متعدد المراحل المداخلة الطريقة لتصميمه في العقد V مرتبة من

1 إلى n من هذه المسألة S يعني الممرس 1 ثم نحسب عقد المجموعة V_2 فهايس متتالية

حتى نصل إلى عقدة النهاية T أي في النهاية المسألة بعد المجموعة $V + 1$ تكون أكبر من

تلك المسألة $V[i]$

```

void PGraph (Graph G ,int k ,int n ,int P[])
{ float cost [maxsize]
  int D[maxsize] ,x ,
  cost[n]=0.0
  for (int j= n-1 ; j>= 1 ; j--)
  { // Compute cost[ ]
    Let x be Vertex Suchthat <j,x> is an edge of G and c[j][x]+cost[x]
    is minimum ,
    Cost[j]= c[ ][x]+ cost[x]
    D[j] = x ,
  }
  P[1]=1 , p[k]=n,
  For(j= 2, j<= k-1, j++)
    P[j]=D[ p[j-1]] ;
}

```

- ونلاحظ هنا أننا نقدر الحواف بدرجة اسم كلتا طرفيها :
 - إن المصفوفة [P] تحوي أرقام العقد الموجودة على المسار ، ونلاحظ أن كل صف من مصفوفة [P] له قيمة واحدة وتسمى تلك القيمة هي مصفوفة [cost] لها العقد x قبل العقد التي هي المرحلة التالية في المرحلة التالية.
 - إن أول عقد نبدأ بها هي العقد الأخيرة تلك هي $cost[n]=0.0$
 - إن إيجاد قيمة x يتم بالاعتماد على j (كله x متساوية لأنها جميعها متساوية) يجب أن تكون أقل من j .
 - والبدء بمخطط Graph ومنه نلاحظ أنه يتم كل شيء C يتم كل شيء.
- المخطط يتكون من مصفوفة مؤشرات إلى بيوت (قد تكون طريقة لتسمية التجارب) كما يلي

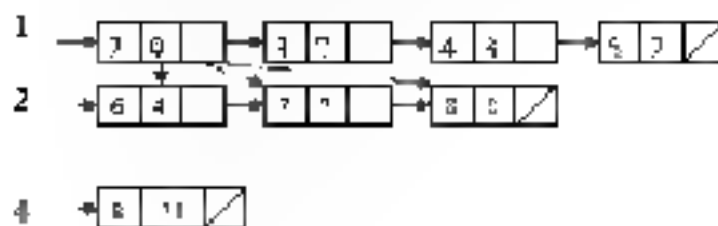
```

Struct node
{
    int vertex;
    float weight;
    Struct node*Link;
}

```

- `vertex` : يمثل بالمعبر ، `weight` : هي القيمة الموجودة على الحافة التي تمثل بالمعبر `weight` ، نضيف هذا الجراء من الجرافيك بمجموعة واحدة فقط
- `Link` : هي المصنوعة لمجموعة بعد فتح كائني

```
Struct node*headnodes[n];
```



headnodes



- لاحظ ان المجموعة الاولى ترابط بطريقة عقد مغروى لذلك استخدمنا `headnodes`
- ان عملية انشاء وفتح الـ `headnodes` يجب ان تتم قبل تنفيذ الخوارزميه `FGraph` ومن ثم يتم استخدامها

وقدنا في جزء البرمجة الخاص بهذه الطريقة التصاعدي (Forward approach)

```

FGraph ( Type head*node )
{
    Type *p=new Type;
    p->Link;
    p->weight;
    p->vertex;
    for(int i=2; i<=m; i++)
    {
        Type *p;
        p->weight;
    }
}

```

```

        p->vertex;
        q->link=p;
    }
    q->link=0;
}
head[u]=0;

void FGraph (Type *head[26],int k,int n)
{ float cost [100],x, int p[100],D[100]
  cost[n]=0;
  for (int j=n-1;j>=1;j--)
    {cost[j]=head[j]->weight+cost[head[j]->vertex],
      D[j]=head[j]->vertex;
      Head[j]=head[j]->link;
      While (head[j] != 0)
        { x=head[j]->weight +cost[head[j]->vertex]
          If (x<cost[j])
            { cost[j]=x,
              D[j]=head[j]->vertex;
            }
          head[j]=head[j]->link;
        }
    }
}

P[1]=1, p[k]=0
for(int j=2;j<=k-1;j++)
  P[j]=D[p[j-1]];
for(int i=1,i<=k-1,i++)
  cout<< p[i]<< " ";
}

```

تعبير بعبارة الدالة (FGraph)

في حالة تنفيذ الخوارزمية القائمة على التكرار (Linked list) فلا يمكن إيجادها في وقت يتناسب مع درجة العقدة (j) (مع ارتباطات العقد مع العقد التي بعدها) (الأسهم) حيث (j) هي من 1 إلى n لذا كل الخوارزمية $O(E)$ أي بحرف قاي وقت بدرجة fox الأولى هو $O(|V| + E)$ حيث V يمثل عدد العقد و E هي تمثل عدد الحواف

هو بالقيمة فوقه حولة f_{opt} النجبه لهر $\Theta(k)$ حيث k يمثل عدد المراحل
هذا يعني ان التكلفة بالقيمة للحواري منه هي

$$\Theta(k) = \sum_{i=1}^k c_i$$

بالإضافة إلى الختر، التي تنطبقه امتدادات "توجد حجة" بوجود خزن التكلفة في المصفوفة
PD

2-6-5: الطريقة التكرارية (Backward approach):

إن تحديد التكلفة هنا يكون من النهاية إلى البداية أي من $(T$ إلى $S)$ حيث هذه الطريقة حلاً
مباشراً وعلاقة حساب التكلفة مع كالاتي.

$$b_{\text{cost}}(i, j) = \min_{\substack{r \in E \\ r \in V}} \{ b_{\text{cost}}(i, j-1, r) + c[r, j] \} \quad \text{حيث } i \in E, j \in V$$

يتم كل التكلفة المرحلية في المرحلة التالية $b_{\text{cost}}(i, j)$ في نفس التكلفة التكلفة التي تربط نقطة
 i إلى نقطة التالية

$$b_{\text{cost}}(i, j) = \begin{cases} \infty & \text{if } i \notin E \text{ or } j \notin V \\ \min_{r \in E} \{ b_{\text{cost}}(i, j-1, r) + c[r, j] \} & \text{if } i \in E \text{ and } j \in V \end{cases}$$

يتم صياغة التكلفة $b_{\text{cost}}(i, j)$ في نفس التكلفة من التكلفة $b_{\text{cost}}(i, j-1)$ إلى التكلفة $b_{\text{cost}}(i, j)$ في المصفوفة b_{cost}
وتكون $b_{\text{cost}}(i, j)$ هي قيمة التكلفة $b_{\text{cost}}(i, j-1)$ بوجود التكلفة $b_{\text{cost}}(i, j-1)$ في نفس
الصفوف b_{cost} ، حيث يتم حساب $b_{\text{cost}}(i, j)$ بقيمة $b_{\text{cost}}(i, j-1)$ في نفس الصفوف b_{cost} بالتسوية بين التكلفة

المصفوفة السبق المرسومة في الشكل (24) فكل التكلفة في أول مرحلة حسابها هي
التكلفة b_{cost} يعني أننا نلاحظ أن التكلفة b_{cost} في الصفوف

$$b_{\text{cost}}(3,6) = \min \{ b_{\text{cost}}(3,2) + 4, b_{\text{cost}}(3,3) + 2 \} = 9$$

بصورة المباشرة على سطح الخرج خطأ أسفل التكلفة التي تعطي القيمة لأحد الصفوف يعني في
الصفوف أو التكلفة المرسومة
والتكلفة التي تعطي القيمة بالنسبة لقيمة التكلفة في أي

$$\begin{aligned} b_{\text{cost}}(3,1) &= 1 \\ b_{\text{cost}}(3,2) &= 10 \\ b_{\text{cost}}(3,3) &= 15 \\ b_{\text{cost}}(3,4) &= 14 \\ b_{\text{cost}}(3,5) &= 16 \end{aligned}$$

$$D \leq \cos f(5,12) = 16$$

أما بالنسبة لإيجاد أقرب من فلز القصور الأقل كلفة من (T إلى S) كان يكلفه مقدار 16
ولتحديد المسار بسجل القصور العشرة في كل حلقة (تعددة) كما يلي

$$D[3,6] = 3, D[3,7] = 2, D[3,8] = 2$$

$$D[4,9] = 6, D[4,10] = 7, D[4,11] = 8$$

$$D[5,12] = 10$$

$$P[1] = 1$$

$$P[4] = 10$$

$$P[3] = 7$$

$$P[2] = 2$$

$$P[n] = 12$$

وباعتبار المسار الأقل كلفة هو

$$S = 1, V_2, V_3, V_4, T = 12$$

حيث

S يمثل العدد الأولي وفي الإثن دائما

$$2 = V(2)$$

$$7 = V(3)$$

$$10 = V(4)$$

12 = T يمثل العدد الأخير

$$V_2 = D[3, D[4,10]] = D[3,7] = 2$$

$$V_3 = D[4, D[5,12]] = D[4,10] = 7$$

$$V_4 = D[5,12] = 10$$

نستنتج إن المسار الأمثل هو : (T) 12, 10, 7, 2, 1(S)

وهذه هي الخوارزمية الشخصية بالخطوط معتمد على حل المسألة بالطريقة الشخصية
(BGraph)

```

void BGraph (Graph G ,int k ,int n ,int 2[])
{ float cost [maxsize],
  int D [maxsize] ,r;
  bcost[1]= 0.0;
  for (int i=2; i<=n ,i++)
  {   Compute bcost[i].
      Let r be Vertex Suchthat <r,i> is an edge of G and bcost[r]+c[r][i]
      is minimum;
      bcost[i]= bcost[r] + c[r][i],
      D[i]= r;
  }
  P[1]=1, p[k]=n;
  For(j: k-1, j<=2 j-)
      P[j]=D[ p[j+1] ]
}

```

و هذا هو جزء البرنامج الخامس بهذه الطريقة التفصيلية

```

FGraph (Type head*node )
{ Type *p=newType;
  q=p;
  p->weight;
  p->vertex;
  for(int i=2; i<=m; i++)
  { type *p;
    p->weight;
    p->vertex;
    q->link=p;
  }
  q->link=NULL;
}
head[n]=0;

void BGraph (Type *head[20],int k,int n)
{ float bcost [100],x; int p[100],D[100];
  cost[n]=0;
  for int j=2 j<=n, j++)
  {bcost[j]=head[j] ->weight+bcost[head[j]->vertex];
    D[j]=head[j] ->vertex;
    head[j]=head[j] ->link;
  }
}

```

```

While (head[j] != 0)
{
    x=head[j]->weight + bcost[head[j]->vertex],
    If (x < bcost[j])
        bcost[j]=x,
        D[j]=head[j]->vertex.
}

head[j]=head[j]->link,
}

}

P[1]=1, p[1]=n
for(int i=b; i<=2; i++)
    P[i]=D[p[i-1]];
for(int i=1; i<=k; i++)
    cout<< p[i] << " ";
}

```

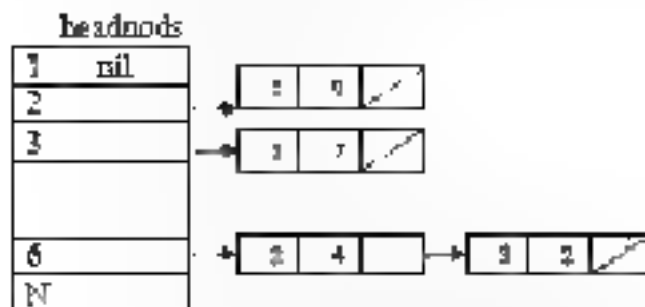
تعيين تعقيداته (EGraph)

في تعقيد الوقت والحرر هذه الخوارزمية هي نفس تلك بتعريف مخزونية (EGraph) ولكن بشرط واحد في تمثيل المخطط ففي تلك الخوارزمية

هذا يعني في التعقيد الكمية مخزونية هي

$$O = |V| + |E|$$

في قولنا الحرر تمثيله تمثيل كالتالي :



في كل عقدة P فليكن W مجموعته أو قيمته W و W هي

E يمثل مجموعته الحرفية ، \forall يمثل الحذف في قائمة بيرس

ملاحظة// هناك تصنيف آخر في التخصصات مبنية للمراحل منها ما يخص الذاكرة (مبنى من العمل يقسم على مجموعته هشاريج)

3-5-3 طريقة القضاء الاثر رجوعا (Back Tracking)

تعتبر إحدى أكثر الطرق البديهية لقصم سحر ابرهناط طب يمكن أكثر من حل للمسألة (مجموعته حول)
ملاحظة // معه proofing نسجم هذا القيد في تنهيا طبعت نبقى كسجه

نفسر معظم المسائل التي تبحث عن مجموعته حلول أو حل امثل يحد بعض القيود
يكون لكل الصيغة (x_1, x_2, \dots, x_n) حيث $x_i \in S_i$ لي مجموعته محددة هي S_i
هذه رئيسية لهذه الطريقة هي انه إذا سركت في الفتحه الجريسي (x_1, x_2, \dots, x_n) من يقود
إلى حل امثل في كل محركات تكوين الفتحهات الجريسي بهس تعاماً

تتطلب العديد من المسائل التي يتخذ حلها باستخدام طريقة القضاء الاثر رجوعاً () أن تحقق
الضرب مجموعته من القيود التي يمكن تقسيمها إلى فئتين وهما مجموعته من القيود
المجموعة التي صريحة (Explicit Constraints) :
وهي قد تعد تقدم S_i المقصود به S_i حيث كل الفتحهات التي تحقق القيود
الصحيحة تكون قراخ للحالات الممكنة

المجموعة التي ضمنية صعيه (Implicit Constraints)
هي هراعد التحديد أي متجهات الضور تحق S_i الهك اي انها نصف ارتباطات S_i بغير ه
الخطوات المستعملة بطريقة القضاء الاثر رجوعاً

- 1 تحديد هراعد الحلول الممكنة للتمهين الإجابي و الما لملئ المسألة
- 2 تقسيم هراعد الأثر S_i بطريقة مبنية للكب (سحره بر مخطط)
- 3 بحث هراعد S_i بطريقة تحقق بوز (Depth_first_Search) مع استعمال دوال تقيد (Bounding Function) لتجنب الحرك إلى هراعد جريسي / نقود إلى حل

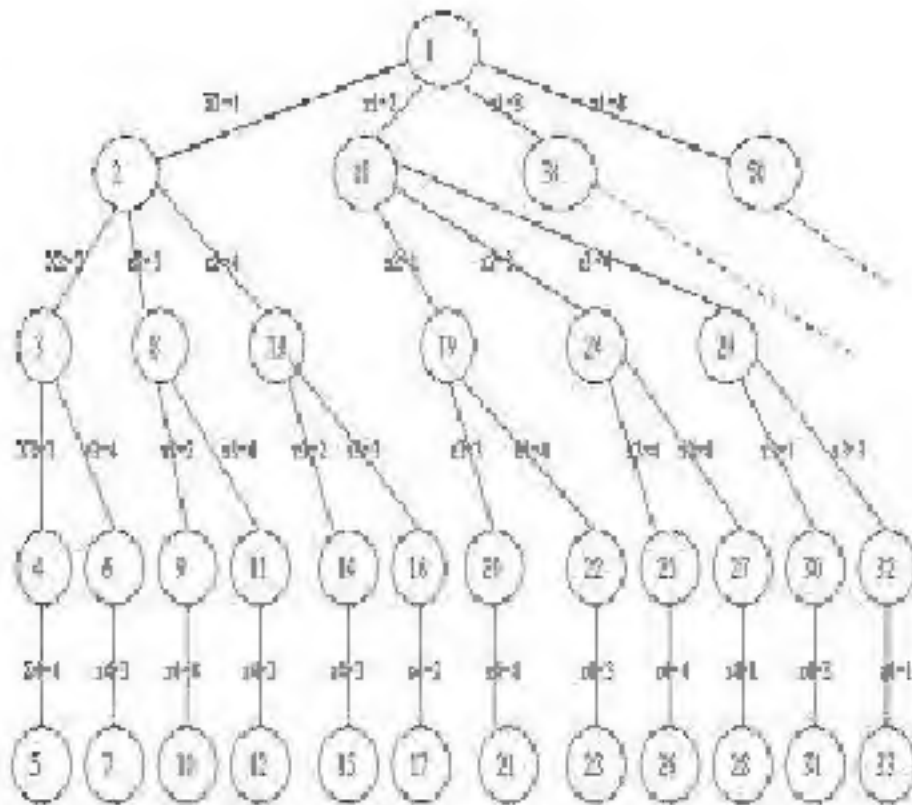
مثلاً : مسأله مملكه n مملكه (n queens problem) كملاً تتحقق هذه الطريقة ، حيث
توضع الملكات على لوحة مخطط بحيث لا تكون هناك ملكتين على نفس الصف أو القطر أو
العمود
ملاحظة : في هذه المسألة يوجد n مملكه من الملكات بوزاد وضعها على لوحة مخطط بعدد n
مملكه لا توضع أكثر من مملكة على نفس الصف أو العمود أو القطر .

لتفرض أن ترتيب صفوف وأعمدة لوحة الشطرنج $[1 \dots n]$ والملكات أيضا $[1 \dots n]$ يمكن وضع الملكة في الصف i ، ستفعل كل طول عملة n ملكة بالنتيجة $X_1, X_2, X_3, \dots, X_n$ حيث X_i هو العمود الذي توضع عليه الملكة i بمعنى $X_i = 2$ فهذا الملكة i توضع على العمود 2.

القيود الصريحة هنا هي $S_1 = [1 \dots n]$

والقيود الضمنية توصف بارتباطات X_i بغيرها
القيود الضمنية هي أن كل الملكة يجب أن تكون على أعمدة وأقطر مختلفة.
القيود الصريحة تعطى متجهيات عند $n = 3$ ، إن القيد الضمني الأول يشير إلى تشكيل من المتجهيات وهذا يفتقر حجم فراغ الحالات أو الحلول من n^3 إلى n^3 .

الآن لدينا 4 صور وعلمنا إيجاد للحلول الممكنة ؟
الخطوة الأولى أن نجد هو 4 أننا يمكن رسم شجرة الاحتمالات (التفصيل) كالتالي :



في هذه العملة يجب أن نراعي الشرط الذي يقول بأن الملكة ممكن أن توضع في العمود الأول أو الثاني أو الثالث أو الرابع كجداية الحل لأن المصفوفة فارغة.

بما إن مجموعة الحلول هي 12 هذا يعني أنه لدينا (24) حالة وإن جزء منها يحقق الحل المطلوب .
 الحراف من قيمة من لكل للقيم الممكنة X_i ،
 الحراف من المستوى i إلى المستوى $i+1$ تحدد قيم X_i بلهذا فإن الشجرة على أقصى اليسار تحتوي على كل الحلول ثلاث X_i تساوي (1).
 إن المسار المتولد من العقد التالية (1,18,29,32,33) يعطي حالة ممكنة يحقق الشرط ، و هو $(2,4,1,3)$ كما في المصفوفة التالية :

	1	2	3	4
1		X1		
2				X2
3	X3			
4			X4	

هذا 12 تقصد بها الملكة الأولى وهكذا بالمتتالية لبقية الملكات .
 أي إن:
 الملكة 1 توضع بالعمود الثاني
 الملكة 2 توضع بالعمود الرابع
 الملكة 3 توضع بالعمود الأول
 الملكة 4 توضع بالعمود الثالث

تمرين // أكتب برنامج يقوم بتطبيق الخوارزمية الخاصة بمسألة (n_queens problem) ؟

References:

- 1-Aho, Alfred V. and Jeffrey D. Ullman [1983]. Data Structures and Algorithms. Addison-Wesley, Reading, Massachusetts.
- 2-Bailer J.:An Introduction to Data Structures ; Allyn and Beacon,Inc,1982.
- 3-Berman A.M.: Data Structures via C++ (objects by Evolution), Oxford University press Inc. ,1997.
- 4-Bertiss A.T. : Data Structures ,theory and practice ; Academic press Inc ,1975.
- 5-Cormen, Thomas H, Charles E. Leiserson and Ronald L. Rivest [1990]. Introduction to Algorithms. McGraw-Hill, New York.
- 6-Dahl O. J. ,Dijkstra E. W and Hoare C.A.R. : Structured 6-programming,Academic press Inc,1972.
- 7-Dale N. and Lilly S.C: pascal plus Data Structures ,Algorithms and Advance programming ,D.C.Heath and company ,1985.
- 8-Goodrich M.T. and Tamassia R. : Data Structures and algorithms in java ;John Wiley and Son Inc. ,1998.
- 9-Gonnet G.H and Beeza -Yates R.:Handbook of Algorithms and data Structures ; Addison_Wesley,1991.
- 10-Horowitz E.and Sahni S: Fundamentals of data Structures in pascal ;Computer Science press Inc,1987.
- 11-Knuth, Donald E. [1998]. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley, Reading, Massachusetts.
- 12-Pearson, Peter K [1990]. Fast Hashing of Variable-Length Text Strings.
- 13-Communications of the ACM, 33(6):677-680, June 1990.
- 14-Pugh, William [1990]. Skip lists: A Probabilistic Alternative To Balanced Trees.
- 15-Communications of the ACM, 33(6):668-676, June 1990.
- 16-Stephens, Rod [1998]. Ready-to-Run Visual Basic Algorithms. John Wiley & Sons, Inc., New York.
- 17-Thomas H. Cormen (Charles E. Leiserson (Ronald L. Rivest and Clifford Stein. Introduction to Algorithms (Second Edition.
- 18-MIT Press and McGraw-Hill, 2001 ISBN7-03293-262-0 .Chapter 1: Foundations, pp.3-122 .
- 19-C.L. PHILIPS& H.T.NAGLE, Digital Control System: Analysis and Design (Prinice- Hall 1984).

- 20-<http://i136.photobucket.com/albums/q175/uraminza/tab1,2,3.jpg>,
- 21-<http://alyaseer.net/files/file.php?id=10>
- 22-<http://akosai.hajznet.com/bubble-sortalgorithm.pdf>
- 23-<http://akosai.hajznet.com/heap-sortalgorithm.pdf>
- 24-<http://akosai.hajznet.com/merge-sortalgorithm.pdf>